UNIVERSIDADE DO MINHO
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ELECTRÓNICA INDUSTRIAL

Desenvolvimento de software para aplicação no controlo e monitorização de plataforma móvel de recolha de bolas de golfe

André Joaquim Barbosa de Oliveira

Dissertação submetida à Universidade do Minho para obtenção do grau de Mestre em Engenharia Electrónica Industrial e Computadores

Guimarães, 2007

Dissertação realizada sob a orientação científica do
Professor António Fernando Macedo Ribeiro,
Professor Associado do Departamento de Electrónica
Industrial da Universidade do Minho.

“O homem é cego aos raios cósmicos, aos raios gama e X, às radiações ultravioleta e infravermelha, às ondas hertzianas do rádio e da televisão. Mas daí a meditação sobre sua grandeza: cego, mas capaz de captar todas essas energias e transformá-las a seu proveito. Cego, mas orgulhoso de seu engenho.”

Harley E. A. Bicas, Marcos P. Ávila

Resumo

Desenvolvimento de software para aplicação no controlo e monitorização de plataforma móvel de recolha de bolas de golfe

Palavras-chave: golfe, software, robótica, processamento digital de imagem, aquisição e transmissão de vídeo, comunicação TCP/IP, JPEG.

O golfe é um desporto que necessita de grande prática e concentração. Para obter tais capacidades são criados campos de treinos onde existem entre outros, o treino de primeira batida designado por *driving range* que permitem ao jogador aperfeiçoar a posição de batida e o alcance máximo da bola. Este tipo de treino necessita de grande *stock* de bolas e um sistema de constante recolha de bolas de golfe simultaneamente à batida das mesmas, o que pode originar perigo para quem recolhe as bolas e elevada despesa para um elevado *stock* de bolas. Para reduzir estes inconvenientes desenvolveu-se uma plataforma móvel de recolha autónoma de bolas de golfe com detecção das bolas localmente, com auxílio de visão computacional.

O presente trabalho consiste no desenvolvimento de software de processamento de imagem para detecção de bolas de golfe, de maneira a direccionar a plataforma móvel para locais de maior concentração de bolas num campo de treinos de golfe. Desenvolvimento de software de monitorização remota do ambiente à volta do robô, através de diversos sensores – implementação de protocolos de comunicação entre três unidades de processamento (unidade de processamento contínuo, unidade de processamento de imagem e unidade de visualização remota).

A metodologia de processamento de imagem usada baseia-se na procura de contornos de uma imagem de escala de cinzentos, com aplicação de filtros de suavização seguidos de filtros de realce de contornos.

Depois de detectados contornos na imagem, são aplicados critérios de selecção dos contornos de modo a seleccionar apenas objectos (bolas de golfe) definidos através de características tais como raio, cor do ponto central, etc. Estas características são obtidas e calibradas através do estudo de várias imagens obtidas no campo de treinos de golfe a diferentes níveis de incidência solar.

Abstract

Software development of an application to control and monitor a mobile platform to collect golf balls

Keywords: golf, software, robotics, digital image processing, acquisition and transmission of video, communication TCP / IP, JPEG.

The golf is an elitist sport that needs great practice and concentration. In order to get such capabilities training camps are created to practice, among others, training of first strike (called driving range) allowing the player to improve ball hit quality and also to achieve long distance balls. This type of training requires a huge stock of balls and a system to constantly collect the golf balls, without stopping the players, which can cause danger to anyone who collects the balls and also a higher budget due to the amount of balls in stock. To reduce these drawbacks an autonomous mobile platform has been developed to collect golf balls where the balls detection is made with the help of computer vision.

This work consists of developing image processing software to detect the golf balls, in order to target the mobile platform for places of higher concentration of balls in a driving range. Software development for remote monitoring of the robot surroundings, through various sensors - implementation of protocols for communication between three units of processing (continuous processing unit, image processing unit and the remote display unit) was the second task.

The image processing methodology used is based on the demand for grayscale image contours, applying smoothing filters followed by highlight of contours.

After detecting the image contours, criteria for the selection of contours in order to select only objects (golf balls) are applied, defined by characteristics such as radius, color of the central pixel, and so on. These characteristics are obtained and calibrated through the study of various images obtained in the driving range at different levels of solar incidence.

Índice

Índice de Figuras

Lista de Tabelas

Capítulo I

Neste capítulo faz-se uma breve descrição acerca da necessidade de elaborar um sistema autónomo de recolha de bolas de golfe, salientando-se os objectivos e possíveis problemas a solucionar. Apresenta-se o método adoptado para a elaboração do projecto e como parte final inclui-se uma breve descrição da solução implementada e uma curta referência à filosofia Open Source adoptada em todo o projecto.

1.1 Introdução

Num jogo de golfe, com o aperfeiçoamento das técnicas de "1ª batida da bola" o jogador consegue alcançar mais rapidamente e com menor número de batidas o objectivo final e consequentemente melhor pontuação. Daí se assumir o treino em campos chamados *driving range* como uma das actividades primordiais num campo de golfe [5]. Este tipo de treino apresenta inconvenientes que se tornam muito dispendiosos – elevado stock de bolas necessário para disponibilizar a modalidade a um elevado número de jogadores e a necessidade de recolha de bolas.

Actualmente a recolha de bolas e sua inserção no sistema dispensador é realizada com intervenção humana e recorrendo ao uso de máquinas com equipamentos atrelados. A necessidade de veículos especializados de modo a melhorar a eficácia do sistema impõe-se, não só para proporcionar uma maior rapidez de todo o processo mas também diminuir os custos de manutenção do sistema.

Os sistemas tradicionais apresentam um desfasamento entre o funcionamento do campo *driving range* e a sua manutenção. É difícil efectuar a recolha de bolas ao mesmo tempo que se encontram jogadores na zona de batimentos – nos casos em que as bolas são recolhidas manualmente é extremamente perigoso para o apanhador andar no campo com vários jogadores a fazer batimentos de bolas de golfe (extrema violência).

Figura 1 Recolha manual de bolas de golfe

Na *Figura 1* salienta-se um sistema tradicional de recolha manual de bolas, onde se pode verificar a necessidade de adoptar medidas de protecção de maneira a diminuir os potenciais perigos para os apanhadores de bolas.

Com o processo de recolha de bolas baseado em veículos motorizados consegue-se diminuir o tempo de espera de recolha das bolas que já foram batidas e diminuir o stock de bolas necessário para o bom funcionamento do campo de treinos. No entanto, ainda existe a necessidade de posicionar o veículo e o seu condutor na zona de impacto das bolas, pelo que persiste a existência de potenciais perigos para o condutor.

O ruído inerente da utilização de motores de combustão é um factor a ter em conta na escolha do sistema de recolha motorizado. Este torna-se extremamente desconfortável para os jogadores que preferem um ambiente de treino pouco ruidoso e com condições favoráveis a uma melhor concentração.

Figura 2 *Recolha de bolas de golfe com veículo motorizado*

Na *Figura 2* é descrito um possível sistema motorizado de recolha de bolas de golfe baseado numa sequência de discos encadeados entre si originando dois rolos que permitem a recolha das bolas de golfe com elevada eficiência, dependendo apenas da perícia do condutor. De salientar a necessidade de adaptar sistemas de protecção idênticos aos da *Figura 1* de modo a diminuir os potenciais riscos para o condutor quando efectua a recolha de bolas com jogadores a fazer batimentos.

1.2 Objectivos e problema a resolver

Pretende-se implementar uma solução que permita diminuir a quantidade de bolas necessárias a armazenar em stock, implementar um sistema móvel autónomo que recolha as bolas de golfe e que permita o correcto funcionamento do campo de treinos *driving range*.

Uma das partes do presente trabalho tem por objectivo principal desenvolver software de processamento digital de imagem que permita o desenvolvimento e aplicação de algoritmos de detecção de bolas de golfe, em campo aberto.

Para que se possa efectuar o controlo e visualização remota dos diferentes sensores é necessário implementar software que permita:

- Transmissão de dados e imagem em tempo real via TCP/IP de modo a monitorizar todo o sistema remotamente;
- Comunicação entre a unidade de processamento contínuo e a unidade de processamento de imagem para que esta controle autonomamente a plataforma móvel de recolha de bolas de golfe.

1.3 Método de resolução do problema

Neste projecto estão envolvidas diferentes áreas da automação e robótica, nomeadamente fusão sensorial, desenho de hardware, sistemas de posicionamento, processamento digital de imagem, accionamento diferencial de motores e mecânica.

Devido à elevada complexidade de todo o projecto, este foi dividido em diferentes vertentes e distribuído por uma equipa de alunos de mestrado integrado.

O processamento digital de imagem é uma parte fundamental de todo o sistema, através do processamento de imagem adapta-se o sistema de posicionamento de modo a efectuar uma rápida detecção de bolas e diminuir o tempo de recolha das bolas de golfe.

O método adoptado para o desenvolvimento do trabalho divide-se em distintas partes:

- Pesquisa bibliográfica e estado da arte;
- Desenvolvimento de software de teste e implementação de algoritmos de processamento digital de imagem;
- Captação de imagens exemplo, que permitam detectar as diferentes características das bolas de golfe e o comportamento da aquisição de imagem em diferentes horas do dia;
- Escolha e teste dos vários algoritmos de selecção de bolas de golfe;
- Aplicação e teste do processamento digital de imagem numa plataforma móvel.

1.4 Solução para o problema

Uma possível solução para melhorar a eficácia de todo o processo – diminuir o tempo de recolha das bolas de golfe, diminuir o número de bolas necessárias por jogador e proporcionar um ambiente pouco ruidoso para os jogadores, é a implementação de um sistema móvel autónomo que:

- Não apresente elevado nível de ruído de funcionamento;
- Recolha o maior número de bolas num curto espaço de tempo;
- Apresente autonomia de funcionamento suficiente para recolher as bolas de golfe;
- Permita um funcionamento autónomo, relativo ao processo de recolher e depositar as bolas de golfe no cais de descarga e iniciar todo o processo de limpeza e fornecimento das mesmas aos jogadores;
- Seja possível estar em funcionamento em horas de grande movimento na zona de batimentos, sem que haja o perigo de danificar a plataforma móvel.

1.4.1 Descrição

De modo a satisfazer todas as condições impostas, desenvolveu-se um protótipo constituído por: um sistema bimotor eléctrico com direcção por diferencial eléctrico, um sistema de visão baseado em aquisição de imagem analógica a cores e um atrelado cuja função é apanhar as bolas de golfe e armazená-las para descarga automática junto à zona de batimentos, quando detectado o máximo de carga.

A *Figura 3* mostra o protótipo da plataforma móvel onde se salienta o sistema de visão com aquisição de imagem analógica e a locomoção com três pontos de apoio – duas rodas com tracção e um terceiro ponto que consiste num atrelado com sistema de recolha e armazenamento de bolas.

Figura 3 *Vista geral da plataforma móvel protótipo*

Para tomada de decisões a plataforma possui uma unidade de processamento contínuo (FoxBoard), que interpreta os sinais vindos de diferentes sensores, das imagens obtidas através da câmara ou dos comandos accionados pelo utilizador através de um *joystick* ou ainda outro dispositivo de acção remota.

A FoxBoard, depois de accionado todo o processo de fusão sensorial, define qual a trajectória a seguir e gera comandos individuais de força para enviar a cada um dos motores, com vista a realizar uma trajectória previamente definida. Esta unidade regista em contínuo toda a informação proveniente dos sensores e actuadores do sistema, sendo por isso capaz de evitar obstáculos, mudar o seu comportamento dependendo das condições atmosféricas e do terreno, informar acerca do estado do sistema incluindo o nível de autonomia das baterias, temperatura e humidade no interior da caixa protectora dos sistemas de processamento.

O atrelado é uma adaptação do sistema de apanhar bolas de golfe, baseado num conjunto de discos de material plástico conforme descrito na *Figura 4*, comercializado pela RbirRangeBalls [6].

Figura 4 *"Mini Hand Push Picker"*

O sistema de visão é constituído por uma unidade de processamento de imagem (pcLocal) com placa de aquisição de imagem através de uma câmara analógica a cores indicada para funcionamento no exterior (CNB-B2310PVF). A visão por computador permitirá detectar localmente, zonas de maior concentração de bolas de golfe.

O sistema de aquisição, processamento de imagem e software de monitorização é o objecto de estudo na presente dissertação, pelo que será apresentado mais à frente em mais detalhe.

1.4.2 Ambiente de programação

Inicialmente desenvolveu-se uma aplicação de teste, em ambiente gráfico *linux* – Ubuntu Desktop Edition [7], de modo a facilitar a percepção das características do terreno e testar o comportamento de determinados algoritmos de processamento digital de imagem em imagens previamente retiradas no campo de *driving range* com bolas de golfe.

O sistema operativo usado para as unidades de processamento da plataforma móvel é *linux* e a linguagem de programação adoptada é *C*. Pelo que foi necessário adquirir novas competências acerca de programação em sistemas operativos linux.

A interface gráfica foi desenvolvida através do recurso à biblioteca *Svgalib* [8]. Esta é uma das bibliotecas disponíveis para criação de grafismos em sistemas operativos sem ambiente de desenvolvimento gráfico. Optou-se pela não utilização de um sistema operativo gráfico, devido à fiabilidade necessária a adoptar em todo o sistema e também devido às limitações de memória de armazenamento e despesas no desenvolvimento do projecto.

O crescente uso de sistemas de visão por computador originou uma vasta gama de bibliotecas de software que simplificam a implementação de algoritmos de processamento digital de imagem. Neste trabalho é usada a versão 0.9.7 da biblioteca *Open Source, Computer Vision Library – OpenCV* [9]. Esta biblioteca apresenta um nível de processamento acessível e de fácil compreensão, que implementa algumas das funções de processamento mais usadas na área da visão por computador, permitindo implementar e testar os objectivos propostos.

1.4.3 Open Source

De modo a diminuir os custos de desenvolvimento e adquirir de forma rápida e eficaz o conhecimento básico necessário para este projecto, adoptou-se uma filosofia de software *Open Source* – uso de software com código fonte disponível para uso comercial ou utilização particular sem qualquer tipo de custo extra, tendo apenas como base o uso de aplicações e bibliotecas testadas e disponibilizadas online. A adopção e uso deste tipo de software permitiram obter soluções para os diversos problemas encontrados ao longo de todo o desenvolvimento, de forma simples e eficaz.

O objectivo primordial da filosofia *Open Source* baseia-se na junção de vários pontos de vista relativos a um mesmo assunto, originando novas ideias e aperfeiçoando metodologias que levam ao melhoramento de soluções.

1.5 Estrutura do documento

Este documento está dividido em seis capítulos distintos. Numa primeira parte faz-se a introdução ao problema a resolver – possíveis problemas e possíveis soluções; seguindo-se um estudo das soluções já existentes e enquadramento teórico das diversas técnicas possíveis de utilizar no processamento digital de imagem. Na parte final apresentam-se metodologias adoptadas, resultados obtidos, conclusões e melhoramentos possíveis de efectuar.

No início de cada capítulo faz-se um resumo do que é referenciado no seu interior.

Ao longo de todo o documento são apresentadas figuras, esquemáticos, tabelas, algoritmos e linhas de código de programação C, de modo a exemplificar e descrever soluções implementadas. Quando se apresentam algoritmos ou linhas de código, estes são definidas por um tipo de letra diferente do corpo do texto normal (`Courier New, 10`).

Capítulo II Estado da arte e enquadramento teórico

Este capítulo começa com uma pequena descrição de possíveis soluções já existentes no mercado, um estudo das características básicas do sistema de visão por computador, estudo de modelos de cor e apresentam-se algumas das funções mais importantes do OpenCV dignas de realce para este projecto.

Na fase final faz-se referência às características da comunicação de dados via TCP/IP – vantagens, desvantagens e descrição de funções de implementação.

2.1 Outras soluções existentes

2.1.1 Sistema com intervenção humana

A maioria dos sistemas de recolha de bolas de golfe disponíveis no mercado, baseiam-se no uso de máquinas com motor de combustão onde é atrelado um reboque tipo o da *Figura 5* que recolhe as bolas para cestos que depois têm que ser descarregados manualmente para a estação de lavagem de bolas.

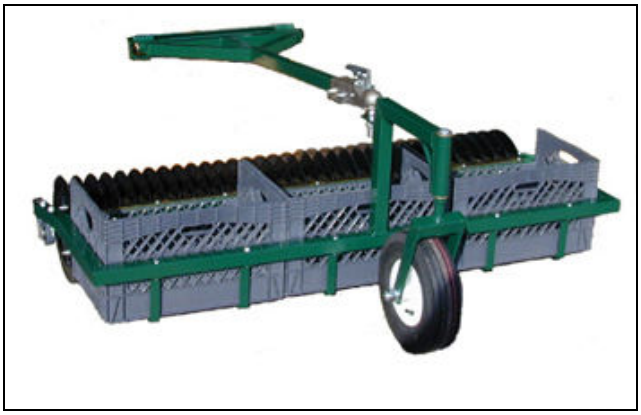


Figura 5 *Sistema de reboque para recolha de bolas de golfe com máquinas de motor de combustão comercializado pela Rangemart*

2.1.2 Sistema autónomo

O robô da *Figura 6* recolhe bolas de golfe autonomamente – BallPicker, *copyright* da BelRobotics [10].

A recolha de bolas é garantida através de varrimento de todo o campo a uma velocidade máxima de 1m/s. Os dispositivos de recolha de bolas baseiam-se num conjunto de discos idênticos aos descritos na *Figura 4*.

Este robô apresenta uma autonomia de funcionamento num ciclo de carga de 2 horas e uma capacidade de armazenamento de até 400 bolas.

A deslocação até a estação de carga de baterias é efectuada de modo autónomo, com detecção de obstáculos através do uso de ultra sons, quando o nível de tensão das baterias atinge o valor mínimo ou quando necessita de fazer descarga das bolas recolhidas – na *Figura 6* está visível o sistema de descarga de bolas e a estação de carga de baterias.

Figura 6 *Robô que apanha as bolas de golfe por varrimento de toda a área, estação de carga de baterias e descarga de bolas*

De modo a delimitar uma zona de trabalho é usada uma "vedação" eléctrica enterrada no solo que define a área interna por onde se quer limitar a passagem do robô. Além do limite por "vedação" eléctrica o robô possui sensores de detecção de contacto (interruptores de contacto) e detecção de obstáculos (ultra sons), que detectam possíveis obstáculos e permitem à plataforma desviar-se atempadamente.

2.2 Processamento de imagem

O processamento de imagem é uma das áreas da automação e robótica que, paralelamente ao aumento da velocidade de processamento dos sistemas computorizados, tem sofrido um grande desenvolvimento.

Paralelamente à aquisição de vídeo tem-se verificado uma simultânea aplicação de algoritmos de compensação de imagem, para melhorar a qualidade da captação de vídeo. Neste tópico apresenta-se uma breve descrição dos métodos de compensação mais usados – Auto Gain Control (AGC), Auto White Balance (AWB), Back Light Compensation (BLC).

2.2.1 Características básicas

É notória uma grande utilização de sistemas de visão em vertentes tais como: detecção automática de defeitos em produtos industriais, interacção homem-máquina, controlo de qualidade em linhas de montagem, identificação automática de objectos, detecção automática de movimento, controlo remoto de ambientes industriais, etc.

Podem-se considerar três tipos de processamento de imagem.

- Processamento baseado na modificação das características da informação visual, com recurso a elementos ópticos – lentes e filtros. Este é um processamento de comum aplicação em fotografia, quer ao nível da revelação quer ao nível da aquisição.
- Modificação do sinal eléctrico que representa a imagem, recorrendo a electrónica analógica.
- Manuseamento digital das características da imagem. Através do manuseamento digital das características dos elementos de informação – pixeis, adquirem-se maior adaptabilidade e flexibilidade dos sistemas automáticos.

Dependendo do tipo de aplicação, os três tipos de processamento podem-se fazer em conjunto, permitindo assim a percepção de características únicas de um objecto.

O processamento digital de imagem é vastamente o mais usado. Este consiste numa sequência de variados passos intermédios. Na *Figura 7* apresenta-se um esquema que elucida as diferentes etapas do processamento digital de imagem segundo *Gonzalez* e *Woods* em *Digital Image Processing* [3] – o início de todo o processo está na aquisição da imagem, que corresponde normalmente à aquisição dos reflexos de luz captados pelo sensor da câmara. Seguindo-se o pré processamento onde se aplicam técnicas de diminuição de ruído, ajuste de contraste e brilho, calibração de câmaras, etc.

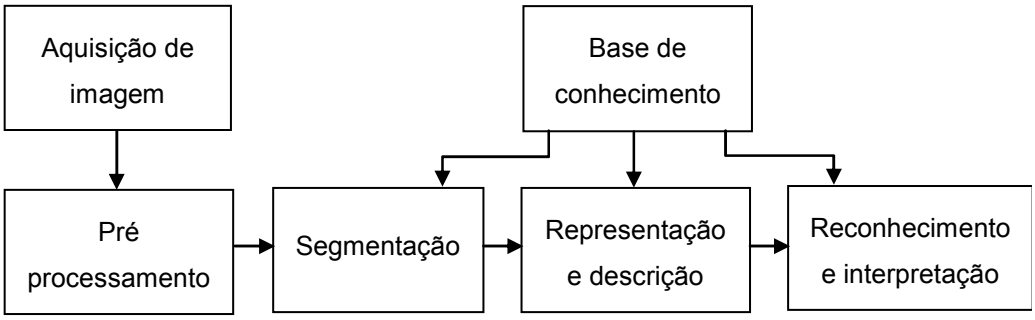


Figura 7 *Etapas principais do processamento digital de imagem*

A *segmentação* define-se como a separação das partes constituintes de uma imagem em diferentes objectos de interesse – daí ser considerada como a parte fundamental de todo o processamento digital de imagem. Nesta fase a segmentação automática é a que se mostra mais eficaz e também mais difícil de alcançar, devido à necessidade dos algoritmos automáticos se terem que adaptar a variações de luminosidade. Uma boa segmentação equivale a uma menor necessidade de aplicar complexos algoritmos de detecção.

O final da segmentação restringe-se, normalmente, a imagens simples. Maioritariamente imagens binárias, que salientam informação relevante dos objectos de interesse tais como contornos e formas ou cores especificas. Estes objectos passam depois pela fase de *representação e descrição* onde se recolhem os parâmetros importantes de cada objecto – área, perímetro, forma, etc.

A última etapa, *reconhecimento e interpretação* pode ou não estar implícita na fase anterior, consoante a complexidade do problema. Através do reconhecimento diferenciam-se, na imagem segmentada, objectos que apresentam características distintas entre eles e pré estabelecidas numa fase anterior ao processamento, que é o estudo dos objectos a detectar. Por exemplo, no caso de detecção de círculos numa imagem, deve-se conhecer previamente qual o tamanho específico dos círculos e se deverão ser

diferenciados consoante a cor predominante – estes dados estão implícitos numa base de conhecimento inerente de uma fase inicial de testes e constatações práticas, que podem estar isoladas de todo o processamento.

A interpretação tem por objectivo juntar um conjunto de informação proveniente do reconhecimento e estabelecer prioridades e/ou tomadas de decisão. Por vezes confunde-se interpretação com reconhecimento, quando apenas se deseja detectar um objecto isolado. No entanto, fazendo analogia ao exemplo anterior, para uma determinada sequência de círculos com determinados tamanhos e cores pode estar inferida uma interpretação distinta. Não bastará apenas reconhecer um círculo, mas sim localizá-lo globalmente em toda a imagem ou sequências de imagens.

2.2.2 Aquisição de vídeo

As câmaras de vídeo analógicas efectuam a captura de vídeo imagem a imagem, espaçadas entre si um tempo predefinido – tempo de captura que define o limite máximo da taxa de *frames* por segundo (fps). Podendo originar uma incoerência entre a definição de aquisição de vídeo e aquisição de imagem.

Algumas câmaras (câmaras de TV ou de uso doméstico) fazem a aquisição da imagem linha a linha, adquirindo um número fixo de linhas espaçadas entre si. De modo idêntico a representação de vídeo nos televisores CRT é baseada no processo de amostragem imagem a imagem, sendo cada imagem composta linha a linha com uma certa frequência de actualização (taxa de reposição de toda a imagem). Este tipo de representação de vídeo funciona correctamente devido ao facto da visão humana não ser sensível a variações de alta-frequência.

Existem dois tipos de câmaras de aquisição de vídeo, as primeiras, e cada vez menos utilizadas, são aquelas cuja captura é efectuada através de tubos electrónicos – alguns exemplos são as câmaras Vidicon, com película foto-condutora de tri-sulfêto de antimónio e as de óxido de chumbo designadas por Plumbicon. Este tipo de aquisição de vídeo apresenta como principais inconvenientes o peso e a inaptabilidade a sistemas de armazenamento digital. As câmaras mais utilizadas actualmente são as que utilizam sensores de estado sólido, nomeadamente as baseadas em dispositivos CCDs – *Charge-Coupled Devices*. Estes dispositivos contêm uma tabela de condensadores, que

interagem entre si através de transmissão de carga eléctrica conforme o comando dado por um circuito electrónico externo [11].

Numa câmara CCD a camada sensível à luz contém um array 2-D de sensores, onde cada um corresponde a um pixel (ou dado) de informação digital. Para capturar as diferentes cores da imagem devem existir três sensores, calibrados para uma gama de frequências predefinidas que correspondem às cores primárias R, G e B respectivamente. De modo a reduzir os custos de produção, a maioria dos produtores de câmaras de aquisição de vídeo opta por incluir num único chip CCD as três componentes de cor dividindo o sensor em três partes distintas, além de um menor custo de produção consegue-se obter um produto final com dimensões menores do que se fosse implementado um sistema baseado em três distintos CCD's [4].

Cada vez mais o processo de aquisição de imagem está afectado pela electrónica analógica/digital. De modo a obter melhores resultados de captação e gravação de imagem sem ajuda externa, os sistemas de aquisição de imagem vêm equipados juntamente com o sensor CCD, com circuitos analógicos e microprocessadores internos que permitem detectar defeitos na imagem adquirida e automaticamente adaptar os tempos de abertura para corrigir saturações de imagem, realçar zonas de menor contaste, eliminar ruídos inerentes de iluminação menos adequada, etc. De seguida apresenta-se um breve estudo de alguns dos circuitos de controlo autónomo.

- **Auto White Balance (AWB)**

Este tipo de controlo faz o ajuste automático da cor consoante o ambiente onde se adquire a imagem. Este é um sistema que normalmente não apresenta boa sensibilidade nocturna pelo que, para usos em ambientes de pouca luminosidade deve-se desligar ou calibrar manualmente o AWB de modo a captar a informação pretendida.

A *Figura 8* mostra uma captação fotográfica com predominância de cor azul, não mostra a cor real dos objectos adquiridos. Na *Figura 9* vê-se a mesma imagem, já com ajuste de *white balance* com cor mais próxima da visualizada pelo sistema de visão humana.

Figura 8 *Imagem sem ajuste white balance.*

Figura 9 *Imagem com auto white balance*

♦ Auto Gain Control (AGC)

O controlo AGC mais básico pode ser um circuito electrónico que detecta a média da cor dos pixeis e quando esta está abaixo ou acima de valores predefinidos, controla o tempo de exposição do sensor originando uma imagem com mais ou menos contraste. De maneira a não permitir comutações simultâneas e para que haja tempos mortos entre os ajustes do tempo de abertura (eliminar o efeito liga-desliga simultâneo) é normal encontrar-se sistemas com transição por histerese.

♦ Back Light compensation (BLC)

A compensação de luz de fundo – BLC, está presente na maioria das câmaras de vigilância e efectua o ajuste do tempo de exposição do sensor CCD para um objecto posicionado numa imagem com luz de fundo forte. É particularmente útil quando se pretende filmar ou adquirir imagens com fundos de tons de cor algo diferentes do objecto a captar, como por exemplo a captura de movimento nocturno ou a detecção de objectos com forte incidência da luz solar.

Existem vários métodos automáticos implementados nas câmaras de vídeo e câmaras fotográficas digitais. É normal proceder-se a um ajuste manual do mesmo de

modo a obter uma imagem com boa definição. Nas câmaras de vídeo é necessário adequar a captura ao objectivo final, caso contrário corre-se o risco do BLC interferir na informação global e apresentar uma imagem de tons de cor elevados (muito clara) ou tons de cor demasiado baixos (muito escura).

Alguns construtores de câmaras de vídeo, como é o caso da *Elbex CCTV systems* [12], optam por não implementar nas câmaras de vigilância este sistema de compensação de luz e incentivar a instalação de iluminação externa, direccionada paralelamente à câmara de frente para o objecto, de modo a realçar o contraste da imagem sem qualquer processamento extra.

A *Figura 10* mostra o resultado da aplicação do BLC. Esta é uma imagem obtida com um processo de selecção manual da zona de processamento, onde se efectuou uma má selecção e não se conseguiu obter informação na zona "backlight" da figura.

Figura 10 *Imagem com má selecção de zona a aplicar BLC*

Na *Figura 11* seleccionou-se apenas a zona de maior claridade e obteve-se a informação relevante da zona da imagem que inicialmente estava saturada. É de salientar o efeito negativo da aplicação do BLC na imagem – o exterior da zona seleccionada fica mais escuro e perde-se informação global obtendo-se apenas bom contraste na zona de maior saturação.

Figura 11 *Imagem com boa selecção de BLC*

2.2.3 Modelos de cor

Algumas das principais desvantagens da visão computacional são: a necessidade de adaptação do software de processamento a um ambiente de iluminação estático e a necessidade de compressão de informação para obter baixos tempos de processamento e, quando necessário, menor largura de banda necessária para transmissão remota de informação.

Sempre que varia a iluminação do ambiente de trabalho onde é efectuada a aquisição da imagem, é necessário adaptar todo o sistema de processamento de modo a detectar de forma correcta todas as características necessárias ao sistema.

Através da adopção e/ou junção de diferentes modelos de representação digital da imagem – espaços ou modelos de cor, consegue-se diminuir o efeito de variação da iluminação.

A maioria dos sistemas usa uma representação de cor baseada no esquema de cores RGB. Isto porque a representação da cor nos sistemas digitais torna-se mais simples – os ecrãs dos computadores e a maioria dos sistemas de visualização digital têm representação de cor aditiva – RGB. No entanto, o sistema ocular humano não obtém uma percepção de cor apenas baseada na junção de três cores primárias, mas sim na base de intensidade (Hue), saturação (Saturation) e luminosidade (Brightness).

Existem várias alternativas de representação da cor, incluindo a PANTONE® Color System, a Munsell Color System, a representação HSV (Hue, Saturation, Value), HLS (Hue, Lightness , Saturation), CMYK conforme referenciado por *Darrin Cardani* [15].

- **Modelo de cor RGB**

O modelo de cor RGB é representado por um cubo. Consoante as coordenadas R, G e B obtém-se uma posição na superfície ou no interior do cubo a que corresponde o valor da adição de cor R+G+B.

A maior desvantagem deste modelo representativo é a total dependência da informação útil da imagem com as três componentes – o que faz com que esta não seja a forma ideal de representar uma imagem quando se pretende obter compressão de imagem.

Na *Figura 12* estão representados os diferentes extremos do cubo RGB, obtidos com a adição das três componentes. De notar a particularidade de todo o plano de escala de cinzentos ser representado com as três componentes de valor igual formando uma diagonal que atravessa todo o cubo desde a origem - preto (0,0,0), ao extremo maior branco (255,255,255) para uma representação a 8 bits.

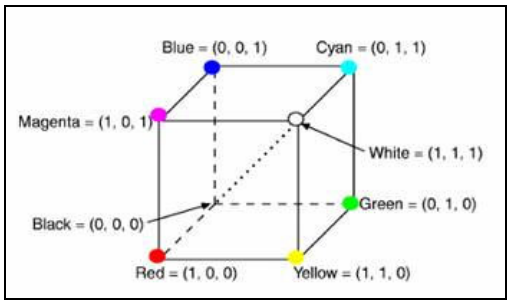


Figura 12 *Cubo RGB com representação das cores nos extremos*

♦ Modelo de cor CMYK

Azul, magenta e amarelo são as três cores secundárias para representação digital (cores primárias para representar pigmentos). Esta representação, ao inverso do modelo RGB é uma representação de cor subtractiva. Ao adicionarmos as três componentes principais de cor, não se obtém branco como no modelo RGB mas sim preto. Este tipo de representação é usualmente aplicada nos sistemas de impressão, daí a existência da componente K para referenciar o preto puro – impressão somente de preto, sem que haja soma C+M+Y tornando o processo de impressão mais barato e não humidifica o papel, etc.

♦ Modelo de cor HSV

O modelo que os utilizadores de sistemas MAC estão mais familiarizados é o sistema de cor HSV – na *Figura 13* é apresentada a representação do cone de cor HSV.

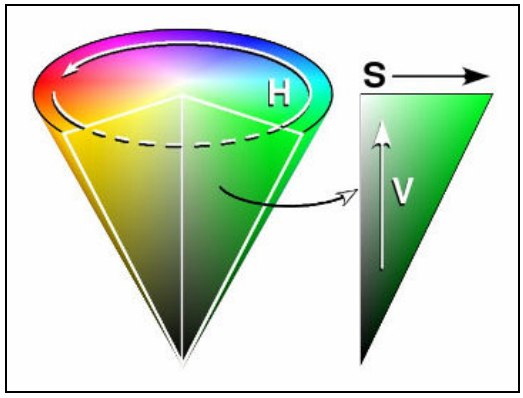


Figura 13 *Cone de cor HSV*

Este modo de percepção da cor é útil no processamento digital de imagem quando se pretende distinguir com alta resolução uma determinada cor. Através do processamento discreto de cada componente consegue-se diferenciar diferentes objectos – é exemplo a detecção de pele humana, com processamento baseado no modelo HSV é possível detectar de modo automático a pele de um sujeito de raça branca ou raça negra sem qualquer processamento extra, apenas processando o canal de cor H.

Outra das vantagens deste modelo de cor é a diminuição dos problemas resultantes de uma iluminação não constante, consegue-se através do espaço de cor HSV uma menor taxa de ruído e uma melhor percepção da informação resultante da imagem adquirida.

- **Modelo de cor YCbCr**

YcbCr (ou YCC) não é considerado um espaço de cor absoluto, é uma codificação da informação RGB de uma imagem. Y corresponde à informação de luminosidade e representa a imagem sem qualquer cor. Cb e Cr representam as componentes de cor azul e vermelho respectivamente.

Este modelo apenas apresenta viabilidade se for usado como conversão do plano de cores standard RGB ou baseando-se numa definição standard imposta a todo o sistema.

Devido à separação das duas componentes de informação – Luminosidade e Cor, e a consequente diminuição de redundância de informação obtida com a representação RGB, este modelo é considerado o que melhor performance consegue introduzir num sistema de transmissão remota de vídeo em MPEG ou JPEG.

Quando existe a necessidade de melhor aproveitamento da largura de banda de transmissão é comum efectuar a transmissão da intensidade Y em alta resolução e as componentes Cb e Cr em menor largura de banda, obtendo uma maior eficiência a todo o sistema [13].

2.2.4 Funções de processamento

- **Binarização**

Este é um dos algoritmos mais básicos de processamento de imagem, muito usado na indústria, devido à sua simplicidade e baixos tempos de processamento para imagens de alta resolução.

Uma das etapas do processamento de imagem é a separação de objectos do seu fundo. O *threshold* (ou binarização) consiste em determinar ou predefinir um limiar

óptimo (*T*), que permita distinguir os objectos numa imagem - em toda a imagem definem-se pixeis com valor acima do limiar com valor lógico 1 e pixeis com valor abaixo do limiar com valor lógico 0 (ou o inverso), conforme representado na equação seguinte.

$$f(x,y) = \begin{cases} 1 & if \ f(x,y) > T \\ 0 & otherwise \end{cases}$$

Neste algoritmo a principal dificuldade está na obtenção do valor óptimo do limiar. Este pode ser obtido experimentalmente e/ou por tentativa e erro. No entanto, cada vez mais se tentam aplicar métodos de detecção automática – pequenas variações da iluminação num sistema de ajuste não automático necessitará de manutenção/calibração.

Segundo *Bryan Morse* [14], uma das técnicas usadas para detecção automática do limiar de transição é o recurso ao estudo do histograma de cor da imagem. Através da procura de vales num histograma de cor consegue-se obter a separação de objectos claros em fundos escuros e objectos escuros em fundos claros.

Na *Figura 14* visualiza-se a possível separação de um objecto do fundo da imagem, através da aplicação de uma binarização com limiar T recorrendo ao histograma de cor.

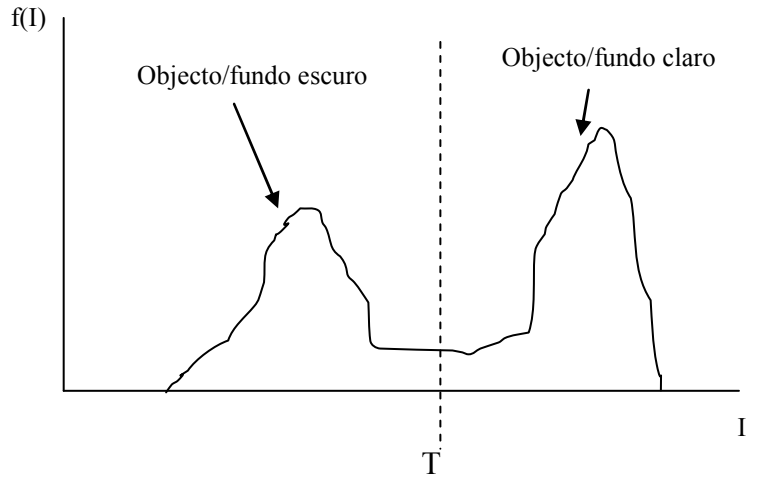


Figura 14 *Limiar automático T obtido através do histograma de cor*

Existem alguns algoritmos de procura que permitem detectar automaticamente um limiar óptimo, como exemplos existem o método de OTSU e o método *adaptive threshold*.

O algoritmo de OTSU procura globalmente um valor que maximize a separação de variância entre classes – objecto e fundo.

O método *adaptive threshold* é um método de procura local que examina estatisticamente a intensidade da vizinhança de cada pixel e devolve um valor de limiar para cada pixel, calculando no final a média desses valores.

Outra das vertentes do threshold é a procura de limiares de transição em segmentos da imagem. Esta consiste em subdividir uma imagem num mosaico de pequenos fragmentos de imagem, determinando-se o histograma de cada sub-imagem. Nos fragmentos de imagem que tenham histogramas bimodais determina-se o valor do limiar de separação apropriado através de métodos, locais ou globais, já referidos atrás. O conjunto de resultados obtidos constitui assim, um conjunto de pontos de referência. Esta metodologia é particularmente útil em aplicações que tenham distintas zonas com diferentes níveis de iluminação.

♦ Equalização de histogramas

Um histograma de intensidades de cor é uma função de acumulação de frequências dos níveis de intensidade que ocorrem numa imagem. Sendo normal efectuar-se a normalização da função dividindo o valor acumulado da intensidade x pelo número máximo de pixeis y.

O objectivo da equalização de um histograma é obter uma distribuição de intensidades de cor uniforme – aumento de contraste em toda a imagem.

O processo de equalização do histograma tem aplicação prática em casos de adaptação automática a diferentes ambientes – zonas de muita luminosidade ou zonas de pouca luminosidade.

Nas figuras abaixo mostra-se o efeito de aplicação da equalização do histograma de uma imagem.

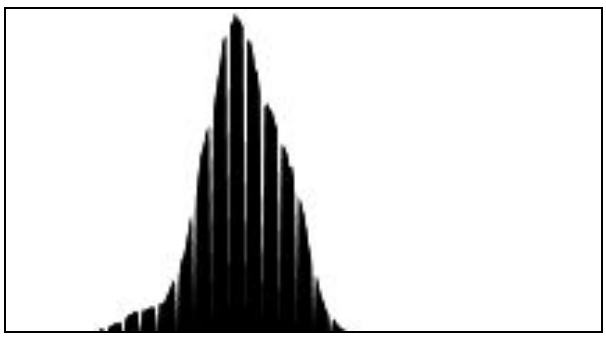


Figura 15 *Histograma sem equalização*

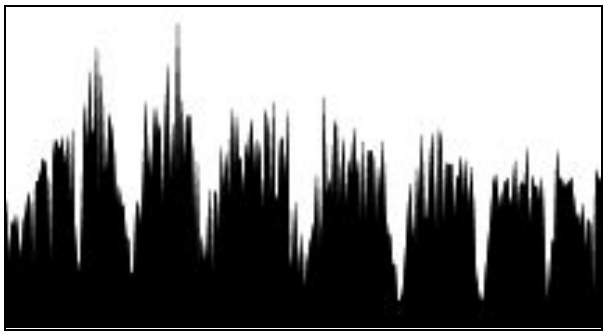


Figura 16 *Histograma com equalização*

Na *Figura 15* a acumulação de frequência está concentrada numa parte do histograma, enquanto que na *Figura 16* a acumulação de frequência distribui-se por todo o eixo horizontal.

Como observação principal, é de realçar que a equalização de histograma não deverá ser efectuada quando se pretende detectar fundos e objectos numa imagem devido ao alargamento de toda a área do histograma e consequente diminuição da zona do "vale" do histograma.

♦ Técnicas de diminuição de ruído

A eliminação de ruído enquadra-se na fase de pré-processamento do sistema de visão por computador. Nesta fase pretende-se eliminar a maior parte do ruído inerente de uma aquisição defeituosa – ruído de aquisição resulta em variações bruscas de tons de cor da imagem, pelo que se torna útil a filtragem de altas-frequências realçando apenas transições lentas. Com a utilização de filtros de suavização – "smooth", obtém-se uma imagem final com suavização de contornos.

O princípio de funcionamento dos filtros digitais é o uso de máscaras (ou kernel) de N dimensão.

Na *Figura 17* apresenta-se um exemplo de máscara de dimensão 3x3. Quando aplicada numa imagem com um factor de divisão igual ao inverso da soma de todos os elementos – 1/9 (para que a média dos tons de cor se mantenha), obtém-se um filtro de média que elimina as altas-frequências, baseando-se na média da vizinhança de cada pixel.

1	1	1
1	1	1
1	1	1

Figura 17 *Kernel 3x3 de filtro passa-baixo*

Um filtro muito usado e com bons resultados na eliminação de ruído é o filtro gaussiano. Este usa uma distribuição normal (ou distribuição gaussiana) para o cálculo dos valores a aplicar no kernel. A equação da distribuição gaussiana de dimensão 2 é dada por:

$$G(x,y) = \frac{1}{(2\pi\sigma^2)} e^{-(x^2+y^2)/(2\sigma^2)} \quad (1)$$

σ é o desvio padrão da função de distribuição de Gauss.

Os pixeis que contêm um valor de distribuição diferente de zero são usados para a construção da matriz de convolução a aplicar na imagem original.

Cada pixel toma o valor da média da sua vizinhança. O pixel original fica com o valor mais alto (ficando com o mais elevado valor gaussiano) e a sua vizinhança fica com valores menores, dependentes da distância ao pixel original.

♦ Detecção de contornos

Entende-se por contorno como o limite de duas regiões. Na detecção de contornos, contrariamente aos filtros de média, pretende-se realçar mudanças abruptas nos níveis de intensidade de cor de uma imagem, pelo que é normal se designar por filtro passa-alto.

Este tipo de segmentação é maioritariamente usado na detecção de objectos.

Um dos métodos mais rápidos (tempo de processamento) de obter contornos de uma imagem é a subtracção do resultado do realce de brancos – através da operação morfológica erosão com a imagem original.

Existem dois tipos principais de detecção de contornos. O primeiro é a detecção baseada em comparações – "template matching", o segundo tipo é a obtenção de

contornos através de uma aproximação ao operador diferencial gradiente (conjunto de primeiras derivadas parciais) da imagem.

- **Detecção de círculos**

A detecção de círculos e objectos arredondados tem aplicação em diversas áreas da visão por computador, nomeadamente na inspecção industrial para detecção de defeitos de produção em série e na robótica em geral para detecção de objectos arredondados.

Um método de correcta detecção de círculos é a aplicação da transformada de Hough (HCT). Através do uso da equação da circunferência e procurando em toda a imagem contornos cujos pontos que o definem pertencem à zona interna definida pela equação, consegue-se obter uma função de acumulação de pontos que mostra quais os centros e raios dos círculos.

$$r^2 = (x - x_i)^2 + (y - y_i)^2 \qquad (2)$$

A maior desvantagem da aplicação da HCT é a necessidade de percorrer toda a imagem e efectuar uma acumulação de "possíveis" centros de círculos. Isto para uma gama de valores de raio que deverá ser definida de modo mais estreita possível para que haja pouco consumo de tempo de processamento.

Outro possível método para a detecção de círculos ou objectos com contornos próximos do círculo é o uso de critérios de área e perímetro do contorno. Embora não seja tão fiável como a HCT, apresenta tempos de processamento muito menores.

2.2.5 Funções do OpenCV

O OpenCV contém uma estrutura denominada `IplImage` que armazena, além de outros dados, a informação da imagem digital – `IplImage.imageData[]`. Esta informação é armazenada de forma a adaptar-se directamente aos sistemas de aquisição de vídeo – representação linear. De seguida apresenta-se o conteúdo da estrutura e respectiva legendagem.

```
IplImage
  |-- int  nChannels;  // Numero de canais de cor (1,2,3,4)
  |-- int  depth;      // Resolucao de cada pixel p.ex. 0-255 8bits
  |-- int  width;      // largura da imagem em pixeis
```

```
  |-- int  height;     // altura da imagem em pixeis
  |-- char* imageData; // represnetaçao linear da imagem
  |-- int  dataOrder;  // tipo de ordenacao de dados
  |                    // 0 - RGB+RGB+RGB
  |                    // 1 - RRRRRR+GGGGGG+BBBBBB
  |-- int  origin;     // origem da imagem
  |                    // 0 - canto superior esquerdo
  |                    // 1 - canto inferior esquerdo
  |-- int  widthStep;  // tamanha de uma linha em bytes
  |-- int  imageSize;  // tamanho da imagem em bytes= height*widthStep
  |-- struct _IplROI *roi;// parte da imagem a ser processada
```

Nas bibliotecas do OpenCV encontram-se variadas funções que permitem aplicar os métodos de processamento descritos em 2.2.4, de modo isolado ou em sequência. Cada função normalmente está definida segundo o padrão `cvMetodo(argumentos)`.

♦ Binarização

A função que se utiliza para efectuar uma binarização é:

```
void cvThreshold(const CvArr* src, CvArr* dst, double threshold,
                 double max_value, int threshold_type);
```

Esta função procura na imagem `src` (1 canal de cor) pixeis com cor maior ou menor (dependendo do `threshold_type` usado) que o valor `threshold` e caso seja maior ou menor, atribui ao pixel, da imagem `dst` de coordenadas iguais, o valor 0 ou `max_value` respectivamente, impondo assim uma representação binária.

Através da função cvThreshold(..) podem-se fazer diferentes tipos de implementação, aplicando os seguintes algoritmos:

```
threshold_type=CV_THRESH_BINARY:
dst(x,y) = max_value, if src(x,y)>threshold
           0, otherwise
```

```
threshold_type=CV_THRESH_BINARY_INV:
dst(x,y) = 0, if src(x,y)>threshold
           max_value, otherwise
```

Estes dois tipos de implementação referidos acima diferem no valor atribuído para valores superior ao limite `threshold` fornecido nos argumentos da função. O método `CV_THRESH_BNARY` atribui valor 255 para cores de valor superior a `threshold` e o método `CV_THRESH_BINARY_INV` atribui valor 255 para cores de valor inferior a `threshold`.

```
threshold_type=CV_THRESH_TRUNC:
dst(x,y) = threshold, if src(x,y)>threshold
           src(x,y), otherwise
```

Para binarização com o tipo `CV_THRESH_TRUNC`, pixeis com cor superior ao valor `threshold` ficam com esse mesmo valor. Pixeis com valores inferiores mantêm-se inalterados.

```
threshold_type=CV_THRESH_TOZERO:
dst(x,y) = src(x,y), if src(x,y)>threshold
           0, otherwise
```

Este tipo de implementação é útil quando se pretende desprezar o processamento de regiões da imagem com baixa luminosidade. Para cores de valor menor que `threshold`, a imagem toma valor nulo (preto).

```
threshold_type=CV_THRESH_TOZERO_INV:
dst(x,y) = 0, if src(x,y)>threshold
           src(x,y), otherwise
```

Esta é uma implementação inversa à anterior – anula regiões da imagem com muita luminosidade, mantendo inalteradas as regiões com valor de cor inferior a `threshold`.

Por vezes não é interessante a aplicação de uma binarização só de um valor, mas sim de uma gama de valores: se A está contido entre X e Y então A=0. Para isso existe a função cvInRange:

```
void cvInRange(const CvArr* src, const CvArr* lower, const CvArr*
               upper, CvArr* dst);
```

Esta função implementa o mesmo algoritmo que a função `cvThreshold(…)`, com a diferença que o valor deverá estar compreendido entre a gama [`lower`, `upper`].

Para pixeis da imagem com valor compreendido entre as constantes `lower` e `upper` a imagem toma valor 255 e zero caso contrário.

♦ Histogramas de cor

Para efectuar o processamento baseado em histogramas de cor, existem funções de criação de histograma e funções de processamento do histograma de uma imagem.

```
CvHistogram* cvCreateHist( int dims, int* sizes, int type,
                           float** ranges=NULL, int uniform=1 );
```

```
void cvCalcHist( IplImage** image, CvHistogram* hist,
```

```
                int accumulate=0, const CvArr* mask=NULL );
```

O processamento do histograma é efectuado a uma dimensão, caso se pretenda estudar/processar o histograma de uma imagem a cores, é necessário implementar rotinas de separação de canais seguidas do processamento discreto canal a canal e por fim juntar toda a informação num só histograma, tendo a principal desvantagem do elevado tempo de computação necessário.

Depois de iniciada a estrutura `CvHistogram`, através da função `cvCreateHist`, e calculado o histograma de cor pela função `cvCalcHist`, o processamento torna-se simples, apenas necessitando de chamar a função `cvEqualizeHist`.

```
void  cvEqualizeHist( const CvArr* src, CvArr* dst );
```

Esta função implementa o algoritmo de equalização do histograma conforme descrito em 2.2.4 e está representado nas *Figura 15* e *Figura 16*.

♦ Diminuição de ruído

O processo de diminuição de ruído, no OpenCV é efectuado através de uma única função:

```
void cvSmooth( const CvArr* src, CvArr* dst, int
               smoothtype=CV_GAUSSIAN, int param1=3, int param2=0,
               double param3=0, double param4=0 );
```

Através da função `cvSmooth` conseguem-se implementar diversos algoritmos de suavização – nomeadamente os algoritmos `CV_BLUR`, efectua a aplicação de máscara `param1xparam2` seguida dum escalonamento de 1/`param1xparam2` e `CV_GAUSSIAN` que aplica uma máscara de dimensão `param1xparam2` com valores obtidos da função normal de distribuição de gauss da eq. 1.

♦ Detecção e processamento de contornos

No OpenCV estão implícitas duas principais funções de detecção de contornos, são elas a função `cvSobel`, que calcula as derivadas de uma imagem através da convolução desta com uma máscara dependente dos argumentos `xorder`, `yorder` e `aperture_size` e a função `cvCanny`.

```
void cvSobel( const CvArr* src, CvArr* dst, int xorder, int yorder, int
aperture_size=3 );
```

```
void cvCanny( const CvArr* image, CvArr* edges, double threshold1,
              double threshold2, int aperture_size=3 );
```

Através da função `cvCanny` consegue-se realçar possíveis contornos na imagem original `image`, e referenciá-los na imagem `edges`. O menor dos valores `threshold1` e `threshold2` serve para estabelecer a ligação entre contornos, o maior serve como ponto de partida para o processo de procura de contornos fortes.

O algoritmo implementado por *J. F. Canny*, em *A computational approach to edge detection* [16], é um dos algoritmos de detecção de contornos com melhor taxa de detecção. Este efectua a procura e detecção de pontos de inflexão da primeira derivada de uma imagem suavizada e estabelece critérios de aceitação de contornos tais como: boa detecção – só ocorre quando existe uma relação sinal ruído baixa; boa localização – quando a média de derivadas da vizinhança do contorno encontrado situa-se próxima do verdadeiro contorno e detecção de somente uma resposta de detecção para cada contorno. Só quando se verificam estes três critérios é que o contorno é considerado válido.

```
int cvFindContours( CvArr* image, CvMemStorage* storage,
                    CvSeq** first_contour,
                    int header_size=sizeof(CvContour),
                    int mode=CV_RETR_LIST,
                    int method=CV_CHAIN_APPROX_SIMPLE,
                    CvPoint offset=cvPoint(0,0) );
```

Esta função implementa a detecção de contornos, guardando as informações relevantes na estrutura do OpenCV `CvMemStorage` e `CvSeq`. É possível escolher o método de aproximação a implementar, bem como o ponto de início de procura.

Para a detecção dos contornos, estão disponíveis diversos métodos de implementação de onde se destacam os modos de aproximação:

- `CV_CHAIN_CODE` – detecção de contornos através do método *Freeman chain code* [18].
- `CV_CHAIN_APPROX_SIMPLE` – compressão horizontal, vertical, e diagonal de todos os segmentos, devolve apenas os pontos extremos.

♦ Detecção de círculos

Para detectar círculos o OpenCV tem uma função dedicada, a `cvHoughCircles`. Esta função implementa apenas um método de detecção (`CV_HOUGH_GRADIENT`), sendo este a conclusão da comparação de resultados obtidos por [17].

```
CvSeq* cvHoughCircles(CvArr* image, void* circle_storage, int method,
                      double dp, double min_dist, double param1=100,
                      double param2=100, int min_radius=0, int
                      max_radius=0 );
```

A detecção de círculos em imagens de escala de cinzentos através da acumulação de valores (raio e centro x_i, y_i da eq. 2) é efectuada pela função descrita acima, sendo necessário apenas definir correctamente a gama de raio dos círculos a procurar, a resolução que se pretende obter nos resultados – resolução com valores elevados origina redundância de detecção de círculos e a distância mínima entre centros de círculos distintos – de maneira a não detectar o mesmo círculo variadas vezes.

Esta função implementa um filtro de detecção de contornos internamente. O que origina uma dificuldade no que respeita à calibração dos parâmetros – `param1` e `param2`.

♦ Conversão de modelos de cor

O OpenCV contém uma função própria que efectua a conversão de uma imagem representada num espaço de cor predefinido para outros espaços de cor de uso normal, simplificando o processo de transição entre modelos representativos implícitos em determinadas placas de aquisição e modelos optimizados para o processamento digital de imagem e/ou compressão de imagem.

```
void cvCvtColor( const CvArr* src, CvArr* dst, int code );
```

Para efectuar a conversão o programador escolhe o modelo de cor origem e o modelo de cor destino, para isso faz-se uso do conteúdo presente no ficheiro `cv.h`, que contém uma lista de definições de códigos para os diferentes tipos de conversão.

Entre os diversos tipos de conversões possíveis salientam-se as seguintes:

- Conversão de 3 ou 4 canais para 4 ou 3 canais de cor RGB ou BGR.
- Conversão de 3 canais de cor para 1 canal de escala de cinzentos.
- Conversão 3 canais RGB para modelo YCbCr, HSV, Lab, HLS, etc.

Todos os tipos de conversão contêm a respectiva conversão inversa de modelo destino-origem.

De seguida apresentam-se os algoritmos de conversão usados para a conversão entre alguns dos modelos de cor descritos em 2.2.3.

Converter RGB para 1 canal (escala de cinzentos) e inverso:

```
RGB[A]->Gray: Y<-0.299*R + 0.587*G + 0.114*B
Gray->RGB[A]: R<-Y G<-Y B<-Y A<-0
```

Conversão RGB para YCrCb e inverso:

```
Y <- 0.299*R + 0.587*G + 0.114*B
Cr <- (R-Y)*0.713 + delta
Cb <- (B-Y)*0.564 + delta

R <- Y + 1.403*(Cr - delta)
G <- Y - 0.344*(Cr - delta) - 0.714*(Cb - delta)
B <- Y + 1.773*(Cb - delta),

            { 128    para imagens de 8 bits,
com delta = { 32768  para imagens de 16 bits,
            { 0.5    para imagens de 32 bits.
```

Conversão RGB para HSV e inverso:

```
V <- max(R,G,B)
S <- (V-min(R,G,B))/V   if V≠0, 0 caso contrário

         (G - B)*60/S,  se V=R
H <- 180+(B - R)*60/S,  se V=G
     240+(R - G)*60/S,  se V=B

se H<0 então H<-H+360
```

O algoritmo acima faz uma representação de $0 \leq V \leq 1$, $0 \leq S \leq 1$, $0 \leq H \leq 360$. É necessária a conversão para o correspondente modelo de cor

```
    Imagens de 8 bits:
        V <- V*255, S <- S*255, H <- H/2 (representação 0..255)
    Imagens de 16 bits:
        V <- V*65535, S <- S*65535, H <- H
```

2.3 Transmissão de dados

A comunicação entre computadores divide-se em dois principais tipos diferenciados – comunicação com conexão e comunicação sem conexão.

Como protocolo com conexão salienta-se o protocolo TCP (*Transmission Control Protocol*). Segundo *Wang* [4], para que haja uma conexão entre computadores deve existir a definição de um cliente e um servidor. O cliente começa por enviar o sinal de sincronismo SYN para o servidor (*Figura 18*), o servidor responde com o envio de SYN seguido de ACK informando a correcta conexão entre os dois e espera pela confirmação por parte do cliente com o envio de ACK, acabado este processo de conexão (ou *three-way handshake*, devido ao envio de três tramas de dados) os dois computadores ficam conectados, prontos a proceder à comunicação de dados.

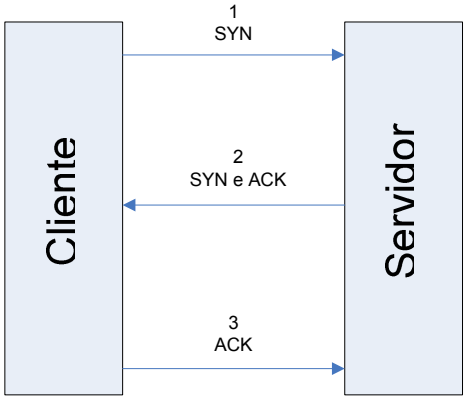


Figura 18 *TCP three-way handshake*

O protocolo TCP apresenta grande fiabilidade e rapidez na transmissão de elevada quantidade de dados, pelo que é maioritariamente utilizado em aplicações de total controlo de informação em que há a necessidade de não perder qualquer byte de informação enviada entre computadores. Na *Figura 19* apresenta-se a sequência de execução padrão das rotinas a implementar na comunicação de sockets TCP.

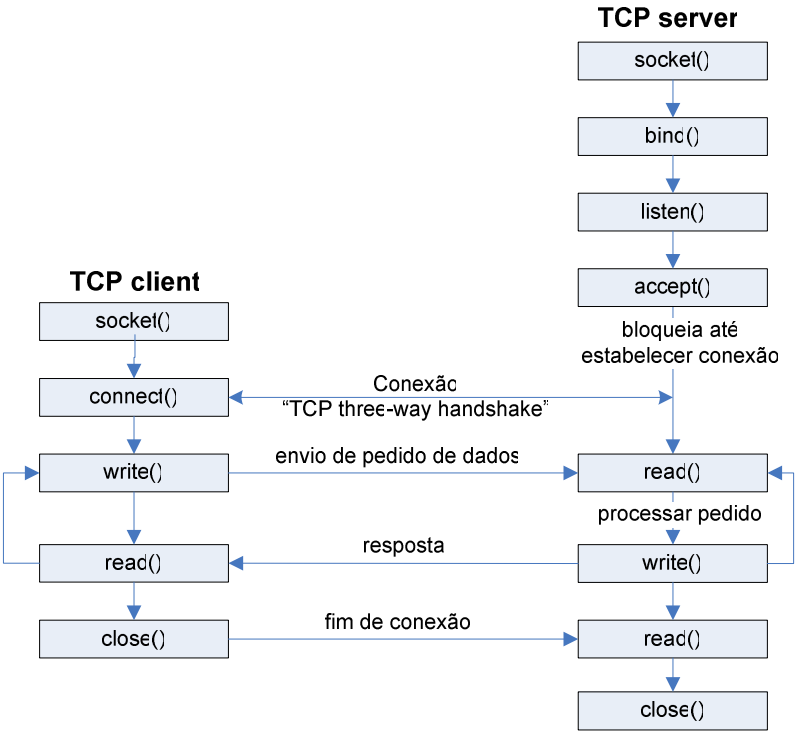


Figura 19 *Funções a executar na comunicação por sockets, para implementar o protocolo TCP [1]*

Um dos problemas provenientes da comunicação baseada em conexões é quando um dos computadores desliga a conexão e há perda de dados na fase intermédia de pedido de "fim de conexão" entre os dois PC's. Isto pode originar a não interrupção da conexão, ficando um dos intervenientes a usar a porta de comunicação sem que haja a noção que esta porta já não está a ser usada. Para a resolução deste problema é comum activar rotinas de controlo de detecção de ligação baseadas em pedido de reconhecimento seguidas de tempo máximo de espera.

Devido ao processo inicial de conexão, a comunicação por TCP sofre de algum atraso inicial, o que pode não ser satisfatório em determinadas implementações daí a existência de um segundo protocolo de comunicação – o UDP (*User Datagram Protocol*). Este protocolo já não implementa um funcionamento baseado em conexões, tornando-o mais simples mas menos fiável – não implementa processos de verificação de envio/recepção de dados. O uso do protocolo UDP é útil na transmissão de dados não críticos ao funcionamento de todo o sistema, como é exemplo a transmissão de vídeo pela internet e monitorização remota de sensores sem qualquer tipo de controlo motor – caso haja uma perda de dados o sistema de comunicação não sofre grandes modificações

ao contrário da transmissão por TCP que, ao detectar a perda de dados tenta restabelecer a conexão inserindo atraso na recepção de dados.

Uma implementação da comunicação de dados via UDP, aplicada por Stevens e Richard em [2], usando rotinas idênticas às da implementação TCP, com a particularidade de serem mais simplificadas e com menor número de bloqueios internos à sequência de execução, está representada na sequência de execução da *Figura 20*.

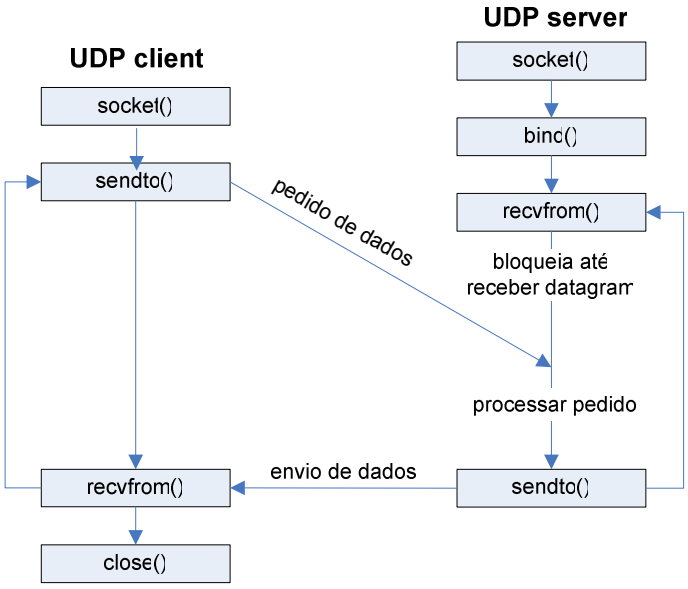


Figura 20 *Funções a executar na comunicação por sockets, para implementar o protocolo UDP[2]*

Depois do processo de iniciação, o sistema servidor espera pela recepção do "datagram" com o pedido de dados na porta de comunicação estabelecida no protocolo de comunicação seguindo-se o processamento do pedido e consequente resposta, caso seja necessário. Pelo lado do cliente, este efectua o envio de um pedido para o servidor e espera pela resposta, apenas fechando a comunicação quando todo o processo esteja finalizado.

Capítulo III Desenvolvimento

O presente capítulo começa por apresentar o processo de instalação de todo o software e hardware necessário para o correcto funcionamento de todo o trabalho. Seguindo-se a referência ao trabalho elaborado, salientando os problemas encontrados e aqueles cuja resolução foi mais morosa – problemas e soluções alternativas encontradas na implementação de algoritmos de processamento digital de imagem, protocolo de transmissão de dados, compressão e transmissão de imagem.

3.1 Instalação de software

Depois de instalado todo o pacote de software básico do sistema operativo linux – Debian no PC de processamento de imagem e Ubuntu Desktop edition no PC de visualização remota do estado de sensores e imagem, procedeu-se à instalação da biblioteca OpenCV e da biblioteca Svgalib e respectiva documentação de apoio – exemplos e tutoriais.

A opção pelos sistemas operativos Debian e Ubuntu deve-se ao facto destes se salientarem de modo positivo entre os diversos utilizadores de sistemas linux, o primeiro como sistema operativo não gráfico e o segundo como sistema operativo gráfico respectivamente.

A biblioteca OpenCV instalada é a versão 0.9.7. Esta versão adequa-se sem qualquer tipo de problemas ao processamento de imagem requerido, pelo que não foi necessário proceder a actualizações.

Para poder utilizar todas as funcionalidades do OpenCV não foi necessário executar qualquer configuração extra, apenas foi necessário fazer a procura de software "OpenCV" nos repositórios opensource dos servidores do sistema operativo debian, através de linha de comandos linux executaram-se os seguintes comandos:

```
    apt-cache search OpenCV; //para procurar software disponível com
referências "OpenCV"
```

```
    apt-get install libOpenCV0.9.7 //depois de uma listagem de software
disponível, optou-se pela biblioteca standard versão 0.9.7.
```

Para a instalação da biblioteca de processamento gráfico Svgalib procedeu-se de modo idêntico à instalação OpenCV, apenas com a diferença de que no final de instalada todo a documentação, houve a necessidade de adaptar o ficheiro de configuração `svgalib.conf` para o tipo de placa gráfica de cada PC, conforme as indicações presentes no próprio ficheiro.

3.2 Inicio da programação

A programação elaborada separa-se em diferentes ficheiros .c e respectivo .h que contém a declaração de variáveis e o cabeçalho das funções existentes no .c, de maneira a distinguir as diferentes rotinas por áreas de implementação e obter uma melhor percepção global de todo o código. Na *Tabela 1* apresenta-se a lista dos diferentes ficheiros implementados e a sua função.

Nome	Área de implementação
Local.c Remoto.c	Funções `main()` – executam a rotina de escolha de opção e contêm as funções de execução dos processos secundários (threads).
geral.c	Funções e variáveis que não se limitam apenas a uma área de implementação
geometrias.c	Desenho de geometrias padrão em SVGALIB
font.c	Desenho de caracteres em SVGALIB
video.c	Controlo e acesso à placa de aquisição
Imagem.c	Processamento e compressão de imagem
rede.c	Comunicação por sockets
sensores.c	Calculo e apresentação de dados dos sensores na interface gráfica

Tabela 1 *Lista de ficheiros utilizados na programação*

Para usar as diferentes bibliotecas e interligar os diferentes ficheiros foi necessário criar um *makefile* que efectua a compilação de todo o programa.

Makefile implementado no pcLocal (unidade de processamento de imagem):

```
# Makefile do GOLFINHO - pcLocal
```

```
CC    = gcc
CFLAGS= -O4 -Wall
LIBS  = -lvga -lhighgui -lcv
OBJ   = geral.o font.o geometrias.o

final: Local

Local: $(OBJ) sensores.o video.o rede.o imagem.o  Local.o
      $(CC) $(CFLAGS) -o Local $(LIBS) *.o -lpthread -lmsock

clean:
      rm -f *.o *~ Local
```

Makefile implementado no pcRemoto (unidade de visualização remota dos sensores e imagem):

```
# Makefile do GOLFINHO - pcRemoto
CC    = gcc
CFLAGS= -O4 -Wall
LIBS  = -lvga -lhighgui –lcv
OBJ   = geral.o font.o geometrias.o

final: Remoto

Remoto: $(OBJ) video.o rede.o imagem.o Remoto.o
      $(CC) $(CFLAGS) -o Remoto $(LIBS) *.o -lpthread -lmsock

clean:
            rm -f *.o *~ Remoto
```

3.2.1 Metodologia de programação

A *Figura 21* descreve o princípio de execução do pcLocal. No inicio do programa faz-se a leitura de dados gerais guardados nos ficheiros de configuração e a criação de todos os processos paralelos (*threads*) necessários para o não bloqueamento da execução do programa principal. Seguindo-se a aquisição da imagem, através da execução de funções de acesso à placa de aquisição de vídeo. Finalizando com o processamento de imagem, que executa algoritmos de detecção da quantidade de bolas visualizadas na zona de passagem do robô e determinar se o robô deve ou não mudar de trajectória. A determinação de uma possível trajectória que permita a recolha do maior número de bolas de golfe baseia-se na detecção de bolas (círculos com centro branco na imagem) e no cálculo do centro de massa de todas as bolas. A direcção fornecida à placa de processamento contínuo é a diferença entre o centro de massa e o centro geométrico da imagem, conforme descrito na eq. 3.

$$direccao = centro\ de\ massa(x,y) - imagem(width,height)/2 \quad (3)$$

$$\text{com}\ \ centro\ de\ massa(x,y) = \frac{\sum_{i=0}^{n} centro(x,y)}{n} \quad (4)$$

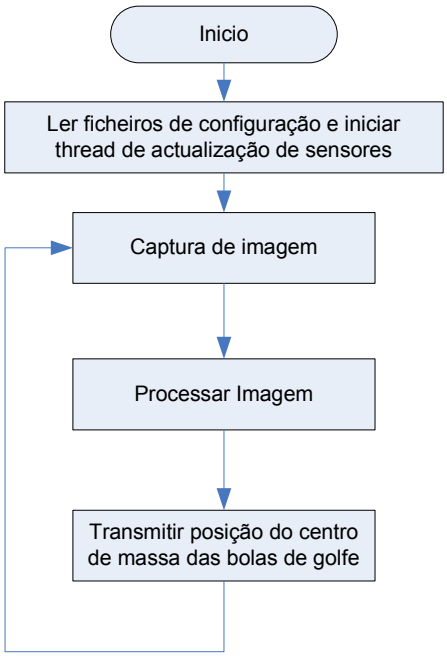


Figura 21 *Ciclo de processamento do pcLocal*

A *Figura 22* apresenta a sequência de execução do programa principal do pcRemoto.

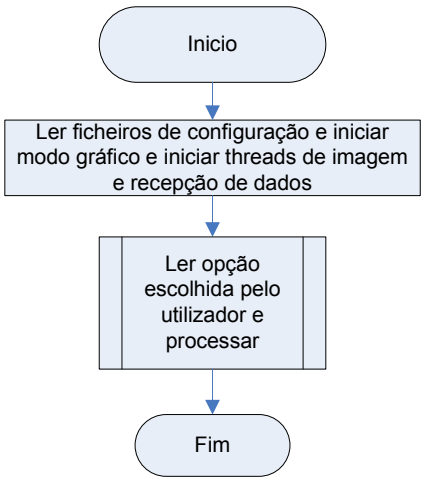


Figura 22 *Ciclo de processamento do pcRemoto*

O programa inicia com a execução do processo de iniciação de todo o software, que de modo idêntico ao programa executado no pcLocal, efectua a leitura dos ficheiros de configuração inicial e inicializa processos secundários que ficam em funcionamento paralelo ao processo principal que comanda as variáveis globais (definidas como *flags*).

Os processos possíveis de execução obedecem ao esquema da *Figura 23*, onde se executam rotinas de funcionamento em paralelo com a rotina principal, são elas:

- Recepção de imagem remota – entra em execução quando se verifica a condição `flagRecebeuImagem==1`, caso contrário fica em estado *standby*;
- Recepção e actualização de dados dos sensores – actualiza os dados dos sensores presentes no robô. Quando é detectada uma trama de dados válida, é activada a variável `flagChegouTrama` que acciona a actualização dos dados dos sensores na interface gráfica.

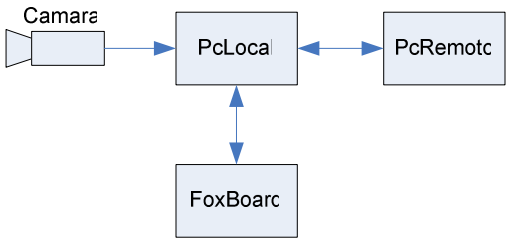


Figura 23 *Esquema representativo das comunicações do software*

O processo `main()` de execução em ciclo infinito só termina quando se verifica a condição `FLAG_SAIR==1` correspondente ao valor da respectiva variável global..

Para estabelecer um padrão entre todo o código e de modo a definir informação relevante e mais usada dentro de todo o sistema, desenvolveu-se uma estrutura que contém os dados que sofrem constantes actualizações durante a execução do programa principal – `struct {…} TRAMADADOS`.

```
typedef struct
{
   char   assinatura[4];        //definir a quem pertencem os dados
   int    bateria12;            // Voltagem da bateria do PC
   int    bateria24;            // Voltagem da bateria da traccao
   float  velocidade;           //velocidade dos encoders
   float  pan, tilt;            // valores do acelerometro
   float  humidade, tempHumi;
   int    bussula;              // de 0 a 255
```

```
   double latitude, longitude;
   int    nSat;                 //nº de satelites que recebe o GPS
   float  tempGps;              //temperatura do GPS
   float  errox, erroy, erroz;  //erro do GPS
   float  velGps;               //velocidade dada pelo GPS
} TRAMADADOS;
```

Esta estrutura, além de permitir o acesso directo a todos os dados globais do sistema, permite uma comunicação entre PC's mais simplificada. Não é necessário estabelecer o envio de cada dado separadamente mas sim transmitir a posição de memória da estrutura `tramaDados` e enviar os `n_bytes` correspondentes ao tamanho alocado na inicialização da estrutura conforme as linhas de código seguintes:

```
enviar_trama((char *)&T, sizeof(TRAMADADOS),2);
```

O código acima efectua o envio da estrutura TRAMADADOS T para o IP destino definido no último argumento.

```
memcpy(&T, (void *)BUFFER, sizeof(TRAMADADOS)); //guarda os dados do BUFFER em T
```

Esta linha de código é a única a executar na recepção de dados, guardando os dados que chegam via TCP na mesma estrutura de dados TRAMADADOS T.

Com estas duas linhas de código e com o uso da estrutura padrão de todos os dados consegue-se uma melhor percepção e comunicação da informação relevante ao sistema.

3.3 Funções de desenho através de SVGALIB

De modo a abstrair a programação das funções internas à biblioteca Svgalib criaram-se rotinas que permitem o desenho de geometrias tais como linhas, quadrados, circlos, etc..

De seguida apresenta-se a lista das principais funções implementadas e respectiva descrição:

Rotina	Descrição
`line(…)`	Linha com inicio em (x,y), fim em (x1,y1)
`linexor(…)`	Linha idêntica à anterior, com a diferença de guardar o desenho de fundo da imagem.
`rectangle(…)`	Rectângulo com inicio em (x,y) e fim em (x1,y1).
`rectanglexor(…)`	Rectângulo idêntico ao anterior, com a diferença de

	preservar o fundo da imagem.
`rectangle_fill(…)`	Rectângulo idêntico ao `rectangle(…)` com preenchimento da área interna a cor.
`circle(…)`	Circunferência de centro (x,y) e raio Z.
`desenha_cruz(…)`	Desenha cruz em (x,y) a determinada cor.
`desenha_cursor(…)`	Apresenta no ecrã uma cruz na posição (x,y) com auxilio da função `desenha_cruz(…)`.

Tabela 2 *Lista de funções de desenho com uso do Svgalib*

A implementação destas rotinas baseia-se no acesso e modificação do conteúdo do endereço de memória gráfica correspondente ao local da imagem (x,y). Todas as funções permitem a selecção da cor da geometria a desenhar.

O uso destas funções é importante no desenvolvimento da interface gráfica, simplificando todo o processo de desenho num sistema operativo não gráfico.

Todas estas funções estão implementadas no ficheiro `geometrias.c`.

3.4 Interface gráfica

Implementaram-se dois tipos de interface gráfica: a interface do pcLocal – funciona como ambiente de *debug*, somente serve para testes e correcção de código no próprio PC; e a interface implementada no pcRemoto – esta funciona como base de visualização remota de tudo o que acontece no robô quando este está em funcionamento. A primeira não vai ser descrita em pormenor, pois é uma adaptação da implementação no pcRemoto.

A resolução usada para a representação da interface gráfica é 640*480 pixeis. Esta resolução revelou-se ideal para o propósito de monitorização do estado dos sensores e imagem, pelo que não foi necessário proceder a grandes modificações a este nível.

Para visualizar o estado de todos os sensores que estão a ser usados implementaram-se rotinas de cálculo, adaptação e desenho dos valores fornecidos pela unidade de processamento contínuo, para poder inserir a informação de modo perceptível na interface gráfica.

Figura 24 *Interface gráfica do pcRemoto, com detecção de bolas e sensores em funcionamento*

Na *Figura 24* apresenta-se a interface gráfica executada no pcRemoto. Com visualização remota da detecção de bolas de golfe – círculos azuis à volta das bolas, e a aquisição de informação de todos os outros sensores nomeadamente, a bússola, o nível das baterias de controlo (12V) e de tracção (24V), o velocímetro, a humidade, a temperatura e a inclinação em dois eixos e a recepção de dados GPS. O mapa do campo (zona a verde) serve para visualização do posicionamento do robô no campo *driving range* conforme a informação fornecida pelo sistema de posicionamento GPS.

3.4.1 Adaptação dos sensores à interface gráfica

Para adaptar os dados de todos os sensores à interface gráfica criaram-se os respectivos ficheiros de programação (`sensores.h e sensores.c`) que contêm as seguintes rotinas:

```
void init_sensores()
{
      DBussola();
      Bateria();
      Velocimetro();
```

```
      Temperatura();
      Humidade();
}
```

A função `init_sensores()` é chamada no início do programa principal e cria o *layout* dos sensores, nomeadamente o desenho da bússola, baterias, velocímetro, temperatura e humidade. Todo o desenho é efectuado sem qualquer referência a valores actuais dos sensores, à excepção das funções `Bateria()` e `Velocímetro()` que já efectuam todo o processo de início e conversão dos dados para desenhar no ecrã.

Sempre que é detectada a recepção de dados, através da variável global `flagChegouTrama=1`, é executada a actualização de todos os sensores através da execução da função `actualiza_sensores()`.

```
void actualiza_sensores()
{
      CBussola();
      Bateria();
      Velocimetro();
      CTemp();
      CHumi();
      CPan();
      CTilt();
}
```

Esta função implementa o cálculo e actualização dos valores da barra de sensores, nomeadamente o cálculo e adaptação da posição da bússola, do nível de tensão das baterias, da velocidade actual do robô, o nível de temperatura e humidade junto ao pcLocal e à FoxBoard e a representação da inclinação do robô.

De seguida apresentam-se as funções de cálculo e adaptação dos dados à interface gráfica em mais pormenor.

`CTemp()` e `CHumi()` são funções de aplicação directa dos valores recebidos, pelo que não necessitam de qualquer conversão, apenas se faz o desenho no ecrã. No desenho do nível de carga das baterias `Bateria()` representa-se o valor em percentagem.

```
void Bateria()
{
int bat=0;

      //12v
```

```
      rectangle_fill(80,410,100,460,corK);     //apagar desenho anterior
      bat=T.bateria12*100/12;                  //percentagem de carga
      rectangle_fill(80,460,100,460-(0.5*bat),corR); //desenhar
      vgaprintf(80,470, corW, "12V");
      vgaprintf(80,400, corW, "%d%%  ",bat);   //escrever valor

      //24v
      rectangle_fill(120,410,140,460,corK);    //apagar desenho anterior
      bat=T.bateria24*100/24;                  //em percentagem 24V
      rectangle_fill(120,460,140,460-(0.5*bat),corG); //desenhar
      vgaprintf(120,470, corW, "24V");
      vgaprintf(120,400, corW, "%d%%  ",bat);  //escrever valor
}
```

`CBussola( … )`

O valor de bússola recebido dos sensores está compreendido entre 0 e 255. Para representar este valor na interface é necessário adaptar o valor inicial a ser representado no interior do círculo e desenhar uma linha a definir qual o posicionamento relativo ao Norte conforme se pode visualizar na *Figura 24*.

```
void CBussola()
{
int i=0;
static float Dg=0.0, y=0.0, x=0.0;

      //Apagar linha anterior com espessura 3
      line( centro_BussolaX,centro_BussolaY,
            centro_BussolaX+x,centro_BussolaY+y,corK);
      line( centro_BussolaX-1,centro_BussolaY,
            centro_BussolaX-1+x,centro_BussolaY+y,corK);
      line( centro_BussolaX+1,centro_BussolaY,
            centro_BussolaX+1+x,centro_BussolaY+y,corK);

      //calculo de valor circular
      Dg=((T.bussula*2*M_PI)/255)-(M_PI/2);

      //calculo de coordenadas (x,y)
      y=sin(Dg)*(raio_Bussola-2);
      x=cos(Dg)*(raio_Bussola-2);

      //Desenho da linha indicadora da bussola com espessura 3
      line( centro_BussolaX,centro_BussolaY,
            centro_BussolaX+x,centro_BussolaY+y,corR);
      line( centro_BussolaX-1,centro_BussolaY,
            centro_BussolaX-1+x,centro_BussolaY+y,corR);
      line( centro_BussolaX+1,centro_BussolaY,
            centro_BussolaX+1+x,centro_BussolaY+y,corR);

      //desenho co circulo do centro, com espessura 3
      for(i=0;i<3;i++)
            circle(     centro_BussolaX, centro_BussolaY, i, corR);
}
```

Conversão do valor linear da bússola (b_l) para o valor circular (b_g):

$$b_g = \frac{b_l.2\pi}{\max\_val} - \frac{\pi}{2} \tag{5}$$

Conversão do valor circular (b_g) para coordenadas (x,y) a aplicar no desenho da linha referencia de direcção da bússola:

$$\begin{aligned} x &= \cos(b_g).raio\_círculo \\ y &= sen(b_g).raio\_círculo \end{aligned} \tag{6}$$

`Velocímetro( … )`

A representação de velocidade (v_l) é efectuada na gama de 0-2m/s e o valor recebido da FoxBoard já está na mesma gama, pelo que apenas é necessário limitar o valor v_l à representação circular definida de $-\pi\ a\ \pi$ e converter através da eq. 6 para coordenadas (x,y):

$$b_g = \frac{v_l.\pi}{\max\_vel} - \pi \tag{7}$$

```
void Velocimetro()
{
int i=0;
static float Dg=0.0, y=0.0, x=0.0;
int centroX=220, centroY=440, raio=25;

     //desenho de circunferencia de espessura 3
     circle(220, 440, 25, corB);
     circle(220, 440, 25+1, corB);
     circle(220, 440, 25-1, corB);

     //apagar meia circunferencia
     rectangle_fill(194,440,246,466,corK);
…
     //calculo do valor circular
     Dg=(T.velocidade*M_PI/2)-M_PI;

     //calculo de coordenadas (x,y)
     y=sin(Dg)*(raio-2);
     x=cos(Dg)*(raio-2);
…
}
```

`CPan( … ) e CTilt( … )`

Estas duas funções calculam e representam a inclinação da plataforma em dois eixos distintos. Para melhor percepção introduziram-se imagens de fundo como referência dos ângulos de visualização do robô e apresenta-se a inclinação através de uma linha obliqua adaptado aos valores de inclinação do sensor, conforme se visualiza na *Figura 24*.

Os valores recebidos (`val`) estão definidos na gama de 0 a 255 e pretende-se uma representação de inclinação de 0 a 360º, efectuou-se a conversão presente na eq. 8. Seguindo-se a conversão para coordenadas (x,y) através da eq. 6 e o desenho de uma recta a passar pelo `centro` de rotação, com inicio em (`centro-x, centro-y`) e fim em (`centro+x, centro+y`).

$$val_g = \frac{val.2\pi}{\max\_val} \tag{8}$$

```
void CPan()
{
static float Dg=0.0, y=0.0, x=0.0;
int centro_PanX=500,centro_PanY=336, raio_Pan=50;

     //apagar linha anterior de espessura 3
     line( centro_PanX-x,centro_PanY-y,
           centro_PanX+x,centro_PanY+y,corK);
     line( centro_PanX-x-1,centro_PanY-y,
           centro_PanX-1+x,centro_PanY+y,corK);
     line( centro_PanX-x+1,centro_PanY-y,
           centro_PanX+1+x,centro_PanY+y,corK);

     //calculo de valor entre 0-360°
     Dg=(T.pan*2*M_PI)/255;

     //calculo de coordenadas (x,y)
     y=sin(Dg)*(raio_Pan);
     x=cos(Dg)*(raio_Pan);

     //desenho de linha obliqua de espessura 3
     line( centro_PanX-x,centro_PanY-y,
           centro_PanX+x,centro_PanY+y,corM);
     line( centro_PanX-x-1,centro_PanY-y,
           centro_PanX-1+x,centro_PanY+y,corM);
     line( centro_PanX-x+1,centro_PanY-y,
           centro_PanX+1+x,centro_PanY+y,corM);
}
```

Nota: A função CTilt() é identica a CPan() com a diferença do valor centro_TiltX=395.

3.4.2 Interacção do utilizador com o software

O software implementado permite a execução de determinadas funções conforme o comando do utilizador, seleccionar uma tecla correspondente a determinada função.

Para ler a opção pretendida foi elaborada uma função geral, que efectua a leitura de teclas e executa a opção correspondente, activando as respectivas variáveis. As funções possíveis estão referenciadas na *Tabela 3*.

Tecla	Opção	Descrição
'4'	Pedido de envio de dados	Envia GOLFD para actualização manual do estado dos sensores.
'1'	Pedido de envio de imagem	Envia GOLFI a pedir o envio da imagem para o pcRemoto.
'p'	Pedido de processamento	Envia GOLFP a pedir o envio de imagem com processamento de imagem.
'c'	Pedido de contornos	Envia GOLFC a pedir o envio de imagem dos contornos, resultante do processamento de imagem.
'P'	Pedido de print-screen	Efectua um print-screen da interface gráfica
'q'	Sair	Fecha todos os processos pendentes e sai do programa.

Tabela 3 *Lista de possíveis opções a escolher na interface gráfica*

A opção de pedido de contornos e pedido de processamento são dependentes do estado anterior do último pedido do mesmo tipo, modificando variáveis presentes no pcLocal segundo o seguinte algoritmo:

```
Se recebeu trama "GOLFP"
        flagProcessamento=!flagProcessamento
Se recebeu trama "GOLFC"
        flagMostraContornos =!flagMostraContornos
```

3.5 Acesso à placa de aquisição de vídeo

A placa de aquisição de vídeo analógico usada é uma adaptação de um sistema comercial de vídeo vigilância, com 4 canais de aquisição.

De modo a simplificar o processo de aquisição de imagem, elaboraram-se rotinas em C que permitem ao programador abstrair-se do tipo de placa de aquisição usada. Estas rotinas estão declaradas, juntamente com as variáveis de manipulação de vídeo, no ficheiro `video.h` e implementadas no ficheiro `video.c`.

`int video_init(void) { … }`

Esta função activa o porto de acesso à placa de aquisição para o modo de leitura e escrita e apresenta todas as características relevantes da aquisição – tamanho máximo de captura, número máximo de canais, etc. É chamada somente uma vez no início do programa principal.

`int video_start(int frame) { … }`

Inicia a aquisição de imagem. É chamada somente uma vez no início do programa.

```
int video_start(int frame)
{
int saida=1;

     my_buf.frame = frame;

     //estabelece sincronismo com a placa de aquisicao
     if (ioctl(fd, VIDIOCMCAPTURE, &my_buf) == -1)
     {    fprintf(stderr, "Error: Grabber chip can't sync INIT\n");
          saida=0;
     }
}
```

`int captura_video(int frame) { … }`

Esta função efectua a aquisição de uma nova imagem. O estado da `flag_captura` define quais os canais que se pretendem efectuar a aquisição (1, 2, 3 ou 4).

O argumento `frame` define qual a imagem a receber (par ou impar).

A aquisição de imagem é efectuada a 640*480*4 canais BGRA, dispostos de modo linear na memória.

```
int captura_video(int frame)
{
int saida=1;
```

```
    //aquisicao do 1° canal
    if (flag_captura&0x01)
    {
         my_buf.frame = !frame;
         if (ioctl(fd, VIDIOCMCAPTURE, &my_buf) == -1)
         {
              fprintf(stderr, "Error: Grabber chip can't sync 1\n");
              saida=0;
         }

         my_buf.frame = frame;
         if (ioctl(fd, VIDIOCSYNC, &my_buf.frame) == -1)
         {
              fprintf(stderr, "Error on sync 1!\n");
         }
    }
}
```

3.6 Comunicação de dados e imagem

Foram implementados dois tipos de comunicação usando o protocolo comunicação UDP. O primeiro é a comunicação de dados entre os três sistemas presentes na rede – pcLocal, pcRemoto e FoxBoard; o segundo é a comunicação de imagem entre o pcLocal e o pcRemoto. Cada tipo de comunicação é efectuado em portas predefinidas – definiram-se portas padrão não pertencentes à gama de portas TCP e UDP reservadas. Para o envio de dados e imagem definiram-se a porta 50000 e 51000 respectivamente.

A imagem enviada remotamente é sujeita a um processo de *resize*, devido à não necessidade de visualização na mesma resolução da imagem processada.

A sequência de execução do processo de envio e recepção de imagem está descrita na *Figura 25* e *Figura 26*.

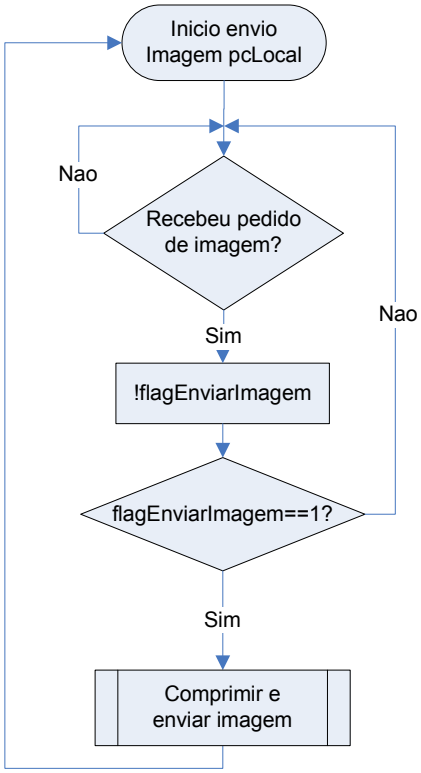


Figura 25 *Sequencia de execução de envio de imagem*

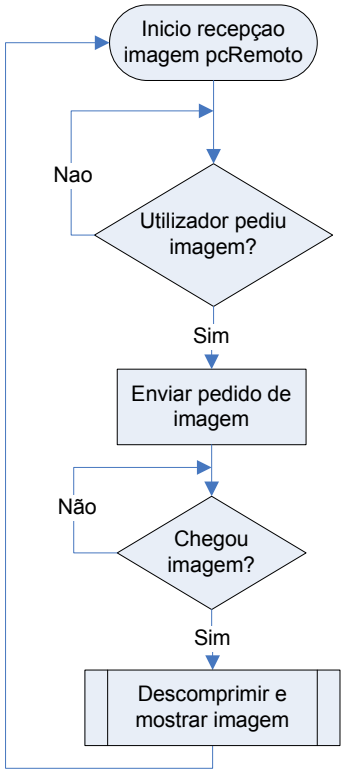


Figura 26 *Sequencia de execução de recepção de imagem*

O envio de dados é efectuado, abrindo e fechando a porta de comunicação do cliente sempre que se efectua uma actualização de dados ou envio de pedido, ficando o

servidor constantemente em processo de *listen* – só fecha a porta de comunicação quando finaliza o programa. O fecho da porta de comunicação do cliente é necessário para libertar a porta no servidor, para comunicação com outra unidade de processamento.

Para a comunicação cliente-servidor criaram-se duas funções, uma para o envio da trama de dados `enviar_trama( … )` e outra para a sua recepção `funcao_servidor( … )`. Ambas as funções implementam uma comunicação por *sockets* TCP/IP.

```
void enviar_trama(char msg[MAXBUF], int number_bytes, int ip)
{
     …

     sock = socket(AF_INET, SOCK_DGRAM, 0); //definicao de socket UDP
     …
     switch (ip) //definicao de endereço ip destino
     {
          case 0: hp = gethostbyname(IP_FOX); break;
          case 1: hp = gethostbyname(IP_PCLOCAL); break;
          case 2: hp = gethostbyname(IP_PCREMOTO); break;
     }
     …
     server.sin_port = htons(PORT); //porta de comunicacao

     //envio de trama de dados
     if ( sendto(  sock, msg, number_bytes, 0,
                   (struct sockaddr*)&server, length) < 0 )
     {
          perror("Sendto");
          printf(" %d Falhou\n",ip);
     }


     close(sock); //fecho de porta de comunicacao em cada envio de trama
}
```

A função de transmissão de dados envia tramas por TCP/IP, referenciadas na descrição da *Tabela 3*, com tamanho e ip de destino definidos nos argumentos. Os destinos possíveis estão definidos na *Tabela 4*.

Ip	Destino
0	FoxBoard
1	pcLocal
2	pcRemoto

Tabela 4 *Lista de IP destino*

A rotina `função_servidor()` é chamada no cabeçalho do programa principal, quando iniciado o processo de comunicação de dados e fica em *standby* até chegar uma trama de dados. Nesta função faz-se a triagem das tramas presentes no barramento de comunicação e activa-se a variável `flagChegouTrama` quando é recebida uma trama com o correcto cabeçalho de identificação.

```
void funcao_servidor(void)
{
     …
     sock = socket(AF_INET, SOCK_DGRAM, 0);     //socket UDP
     …
     //entra em ciclo infinito ate ao fim do programa principal
     while (flagThreadServidor)
     {
          //recebe dados do barramento TCP/IP
          bytes_recv=recvfrom( sock,BUFFER,MAXBUF,0,
                               (struct sockaddr*)&from,
                               (socklen_t *) &fromlen);
          …
          if (strncmp(BUFFER, "GOLF", 4)) //verifica cabecalho da trama
          {
               conta_TRAMAS_LIXO++;
               flagChegouTrama=-1;
          }
          else
          {
               if(BUFFER[4]=='I') //pedido de imagem
                     flagEnviarImagem=!flagEnviarImagem;
               else
                     if(BUFFER[4]=='D') //pedido de actualizacao de dados
                          flagEnviarDados=1;
                     else
                          if(BUFFER[4]=='C') //activar contornos
                          flag_mostrarContornos=!flag_mostrarContornos;
                          else if(BUFFER[4]=='P') //activar processamento
                                flag_processar=!flag_processar;
                          else //chegaram dados da FoxBoard
                          {
                                flagChegouTrama=1;
                                …
                          }
          }
     }
     close(sock); //fecha a porta quando terminar o programa
}
```

3.6.1 Transmissão de dados

Para poder visualizar o comportamento do robô remotamente e interligar as unidades de processamento – unidade de controlo contínuo, unidade de processamento de imagem e unidade de acesso remoto, desenvolveu-se um protocolo de comunicação

Servidor-Cliente conforme ilustrado na *Figura 27*. Este protocolo tem como servidor principal a unidade processamento contínuo (FoxBoard), funcionando a unidade de processamento de imagem (pcLocal) como intermediário entre a FoxBoard e a unidade de acesso remoto (pcRemoto).

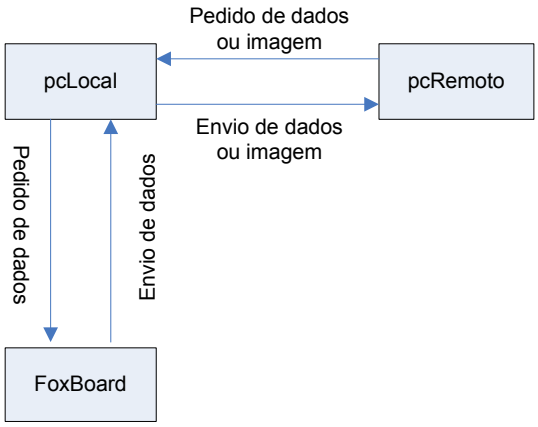


Figura 27 *Protocolo de comunicação entre unidades de processamento*

O protocolo de transmissão foi definido da seguinte forma:

- A FoxBoard deverá ser o 1º sistema a ligar-se e ficar no estado de *listen*, à espera de pedidos de actualização/envio de trama, que contém os dados necessários para a monitorização de todo o sistema, efectuado pelo pcLocal.
- O pcLocal funciona como servidor para o pcRemoto e como cliente para a FoxBoard – sempre que é enviado remotamente um pedido de dados o pcLocal envia os respectivos dados. No caso do pedido ser uma imagem é iniciado o envio remoto da imagem adquirida no robô, finalizando o envio somente quando recebe nova trama de pedido de imagem (um pedido de imagem inverte a variável `flagEnviarImagem` do pcLocal).

Na *Figura 28* apresenta-se a composição básica da trama de dados imposta pelo protocolo de comunicação.

GOLF	**Tipo**	**Dados**

Figura 28 *Trama de envio/recepção de dados*

Toda a trama de pedido ou envio de dados tem o cabeçalho "GOLF" que a identifica como pertencente ao sistema, contém um conjunto de bytes que indica o tipo

de dados que vêm descritos no resto da trama ou o tipo de pedido que está a ser efectuado. O pedido de dados é de dois tipos – pedido de actualização de sensores e pedido de envio de imagem para pcRemoto. Quando é pedido o envio de imagem, o pcRemoto não recebe nenhuma trama com cabeçalho GOLF, simplesmente fica à espera que cheguem dados na porta UDP predefinida para a recepção de imagem.

3.6.2 Transmissão de imagem

O envio remoto de imagem é efectuado num processo paralelo ao processo principal – é criado um novo processo para a execução da rotina `função_imagem()`.

A transmissão de imagem inicia quando é recebida no pcLocal a trama "GOLFI". Ao receber esta trama, é activada a `flagEnviarImagem` dando inicio ao envio de imagem para o pcRemoto através da função `enviar_imagem()`. A transmissão pára quando é recebida nova trama "GOLFI", desactivando a `flagEnviarImagem`.

```
void funcao_imagem(void)
{
unsigned char shrink[MAX_TAM_JPEG];
int tamanho_mem=0;

     init_envia_imagem();
     while(1)
     {
          if(flagEnviarImagem)
          {
               //comprimir imagem
               tamanho_mem = Imagem_comprimir(ipl_shrink, shrink);

               //enviar imagem comprimida
               envia_imagem(shrink, tamanho_mem);
          }
     }

     //termina o processo quando fechar o programa principal
     close_envia_imagem();
}
```

Para a correcta recepção de imagem e melhor taxa de transmissão, foi necessário separar o envio de imagem em três funções distintas: a rotina `init_envia_imagem()` é executada no cabeçalho da `função_imagem()` para abrir a porta UDP de comunicação entre cliente e servidor e estabelecer a conexão entre as duas entidades.

```
void init_envia_imagem( void)
{
struct sockaddr_in      servaddr;
```

```
     bzero(&servaddr, sizeof(servaddr));
     servaddr.sin_family = AF_INET;
     servaddr.sin_port = htons(SERV_PORT);
     inet_pton(AF_INET, IP_PCREMOTO, &servaddr.sin_addr);

     socket_imagem = socket(AF_INET, SOCK_DGRAM, 0);

     connect(socket_imagem,(      struct      sockaddr      *)&servaddr,
sizeof(servaddr));
}
```

As rotinas `envia_imagem()` e `close_envia_imagem()` permitem o envio remoto de imagem, efectuando o fim da conexão e o fim do programa principal de modo simultâneo.

3.6.3 Recepção de imagem

Em sincronia com o processo de transmissão de imagem criou-se a `funcao_imagem()` no pcRemoto.

```
void funcao_imagem(void)
{
     …
     init_recebe_imagem(); //abrir porta UDP e estabelecer conexao
     while(1) //termina juntamente com o programa principal
     {
          if (pedeVideo )
          {
               enviar_trama( trama, strlen(trama), 1);
               pedeVideo=0;
          }

          //receber imagem comprimida JPEG
          recebe_imagem(img_comprimida, MAX_TAM_JPEG) );

          //descomprimir JPEG para RGB
          Imagem_descomprimir(img_comprimida, ipl_img3ch);

          //converter RGB para BGRA, para poder apresentar no ecra
          cvCvtColor( ipl_img3ch, ipl_img4ch, CV_RGB2BGRA);

          //mostrar no ecra a imagem recebida
          copytoscreen_old(ipl_img4ch->imageData, 0, 0);
     }

     //libertar memoria
     cvReleaseImage(&ipl_img3ch);
     cvReleaseImage(&ipl_img4ch);

     //fechar comunicacao
     close_recebe_imagem();
}
```

Esta função implementa a recepção de imagem através da rotina `recebe_imagem()`, que efectua a chamada da função de recepção de dados da programação por *sockets*, seguindo-se a descompressão, conversão e amostragem da imagem recebida.

3.7 Rotinas de processamento de imagem

O processamento de imagem implementado segue a sequência descrita na *Figura 29*.

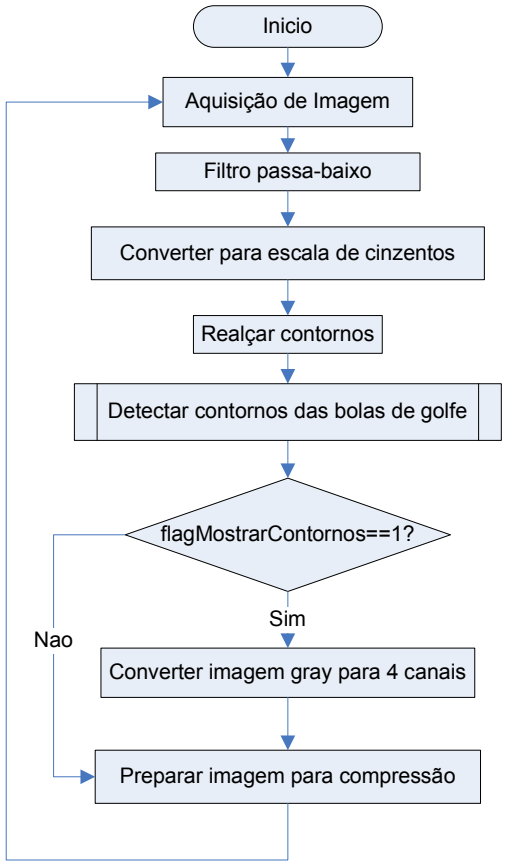


Figura 29 *Sequencia de execução de processamento de imagem*

Inicia-se com o processo de filtragem de ruído através da função `cvSmooth(img, img, CV_BLUR, 3, 0, 0 )`, faz-se a conversão da imagem adquirida na câmara analógica (formato BGRA) para 1 canal de escala de cinzentos através da função `cvCvtColor( img, gray, CV_BGRA2GRAY)` – esta função executa algoritmos de conversão predefinidos na biblioteca OpenCV.

Segue-se o realce de contornos através da função `cvCanny( gray, gray, 325, 400, 3)` – os parâmetros usados foram definidos depois de um processo de calibração manual, adaptado a vários tipos de iluminação possíveis de encontrar no ambiente de trabalho do robô (diferentes horas do dia).

Para a detecção de contornos de bolas de golfe implementou-se a rotina `Contours(IplImage *img, IplImage *img_gray)`:

```
void Contours(IplImage *img, IplImage *img_gray)
{
…

//deteccao e organizacao de contornos
cvFindContours(g, storage, &contours, sizeof(CvContour),
               CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0) );
…
      while(contours)
      {
            …
            //deteccao de características do contorno
            cvMinEnclosingCircle( contours, &centro, &raio);
            perimetro = cvArcLength( contours, CV_WHOLE_SEQ, -1 );
            area = fabs(cvContourArea( contours, CV_WHOLE_SEQ));
            c=(perimetro*perimetro)/area; //c=P*P/A=4*pi para o circulo

            //definir gama de raio
            if(   raio>=4 && raio<9
                  //testar se é um circulo
                  && c>=4*M_PI-tolerancia && c<=4*M_PI+tolerancia
                  //excluir extremos da imagem
                  && centro.x>5 && centro.x<img->width-5
                  && centro.y>5 && centro.y<img->height-5

                  //testar cor do centro do contorno RGB
                  &&(unsigned char) img->imageData[(int)centro.x*4+
                    (int)centro.y*widthvideo*4]>254
                  &&(unsigned char) img->imageData[(int)centro.x*4+
                    (int)centro.y*widthvideo*4+1]>254
                  && (unsigned char) img->imageData[(int)centro.x*4+
                    (int)centro.y*widthvideo*4+2]>254 )
            {
                  countBolas++;
                  centroMassa.x+=centro.x;
                  centroMassa.y+=centro.y;
                  …
            }
            contours = contours->h_next; //passar ao próximo contorno
      }

      //calcular centro de Massa das bolas detectadas
      centroMassa.x=centroMassa.x/countBolas;
      centroMassa.y=centroMassa.y/countBolas;
      …
}
```

Esta função efectua a detecção de contornos através do uso da função `cvFindContours()` e implementa a metodologia de detecção de bolas de golfe referida em 3.7.1.

Para a selecção dos contornos correspondentes às bolas de golfe efectua-se uma procura baseada nas características próprias da representação de uma bola de golfe numa imagem, nomeadamente o raio e centros da bola – calculados através da função `cvMinEnclosingCircle( contours, &centro, &raio`), esta função define um raio mínimo da circunferência que abrange todo o contorno e devolve além do raio o centro geométrico do contorno. Se os valores dos contornos encontrados estiverem dentro de determinados limites estudados, esse contorno pode ser considerado um objecto de procura. Para ser totalmente definido como bola de golfe ainda é necessário testar se a razão entre perímetro e área do contorno correspondem à constante C, que define uma circunferência, conforme descrito na eq. 9.

Através das funções `cvArcLength( contours, CV_WHOLE_SEQ, -1 )` e `cvContourArea( contours, CV_WHOLE_SEQ)` detectam-se os valores de perímetro e área respectivos, faltando apenas comparar a cor central do contorno e testar se esta está dentro da gama de brancos aceitáveis como bola de golfe.

Quando se verifica a condição `flagMostrarContornos=1` (foi activado o pedido de imagem com apenas a detecção de contornos) converte-se a imagem de escala de cinzentos para representação na interface gráfica em 4 canais BGRA cvCvtColor`( gray,img, CV_GRAY2BGRA)`. Caso não seja activada a opção enviar contornos, a imagem a transmitir por comunicação sem fios já está no formato de 4 canais.

Para proceder à compressão da imagem enviada para o pcRemoto, é necessário reduzir as suas dimensões para 320*240, aplicando a função `cvResize( img, ipl_shrink, CV_INTER_LINEAR )`.

```
if(flag_mostrarContornos)
    cvCvtColor( gray,img, CV_GRAY2BGRA);

//captar a 640*480 e reduzir para 320*240 para transmissão de imagem
cvResize( img, ipl_shrink, CV_INTER_LINEAR );
```

3.7.1 Metodologia de detecção de bolas de golfe

A procura de bolas de golfe é efectuada baseando-se em três princípios:

- O contorno de uma bola tem uma forma cuja relação geométrica C, deve ser igual ao resultado da eq. 9.

$$C = \frac{P^2}{A} = \frac{(2.\pi.r)^2}{\pi.r^2} = 4.\pi \tag{9}$$

Sendo A = área, P = Perímetro e r = raio do circulo.

- Toda a bola tem uma predominância de cor próxima do branco puro – assume-se como uso normal apenas bolas de golfe brancas.
- O fundo na imagem de processamento (o relvado) tem uma cor e padrão distinta das bolas de golfe.

Aplicando estes princípios elaborou-se o seguinte algoritmo de procura de bolas de golfe:

```
Para todos os contornos
      Se constante C=4π +- tolerância
            Procurar círculos através da constante C
            Para cada círculo detectado
                  Se cor do pixel central = branco +- tolerância de cor
                        Incrementar contador de bolas
```

3.7.2 Compressão de imagem

Nas comunicações de dados e imagem existe limite de largura de banda de transmissão, o que obriga a uma compressão dos dados para uma rápida e correcta transmissão.

No envio remoto de imagem, efectua-se a compressão e descompressão de imagem de modo a diminuir o envio de dados redundantes.

A biblioteca de compressão de imagem usada foi a JPEGLIB JFIF. De entre as várias rotinas nela implementadas existem duas rotinas que foram usadas no software implementado:

```
write_JPEG_memory (unsigned char *img_buf, int width, int height,
                   unsigned char *jpeg_buffer,
                   unsigned long jpeg_buffer_size,
                   int quality, unsigned long *jpeg_comp_size)
```

Esta função é usada na fase inicial de envio de imagem para o pcRemoto, permitindo a conversão directa de imagem linear de 3 canais RGB para JPEG. Como argumentos insere-se a estrutura que contém os dados da imagem com representação linear, a largura e comprimento da imagem, o tamanho de memória máximo permitido após a compressão da imagem para JPEG, a qualidade (em percentagem) da compressão efectuada e a variável a guardar o tamanho final da compressão respectivamente.

```
read_JPEG_memory ( unsigned char *img_buf, int *width, int height,
                   const unsigned char *jpeg_buffer,
                   unsigned long jpeg_buffer_size);
```

Após receber a imagem por TCP/IP executa-se esta função, que efectua a conversão directa de JPEG para imagem linear de 3 canais RGB.

Como argumentos insere-se a estrutura que guarda os dados da imagem linear após descompressão, a largura e comprimento da imagem, o endereço de memória de início da imagem JPEG e o tamanho de memória máximo permitido após a compressão da imagem para JPEG respectivamente.

Para executar a compressão da imagem implementaram-se duas funções distintas:

```
int Imagem_comprimir(IplImage *orig, unsigned char *dest)
{
     unsigned long tamanho_jpeg=0; //tamanho da compressão
     IplImage *orig_;              //imagem auxiliar

     //inicializa imagem auxiliar
     orig_=cvCreateImage( cvGetSize(orig), 8, 3);

     //converte imagem original BGRA para imagem auxiliar RGB
     cvCvtColor( orig, orig_, CV_BGRA2RGB );

     //inciar compressao de imagem
     write_JPEG_memory ((unsigned char*)orig_->imageData,
                        orig_->width, orig_->height,
                        &dest[0], MAX_TAM_JPEG, QUALITY_JPEG,
                        &tamanho_jpeg);

     cvReleaseImage(&orig_);

//devolve tamanho da compressao
return (int)tamanho_jpeg;
}
```

Esta função converte a imagem `orig`, com representação de 4 canais BGRA, em representação linear com compressão JPEG guardada em `dest` e devolve o tamanho obtido no final da compressão. Foi necessário criar a imagem auxiliar `orig_` de modo a não manipular os dados originais com a conversão de RGB para BGRA no final da rotina.

A descompressão dos dados recebidos por TCP/IP no pcRemoto, para representação linear RGB e retorno do tamanho da imagem recebida é efectuado através da função `Imagem_Descomprimir()`.

```
int Imagem_descomprimir(unsigned char *orig, IplImage *dest)
{
     int w,h;

return (read_JPEG_memory ( (unsigned char*)dest->imageData,
                           &w, &h, &orig[0], MAX_TAM_JPEG));
}
```

Capítulo IV Testes e Resultados

4.1 Aquisição de imagem

Numa primeira fase optou-se pela aquisição de imagem proveniente de três câmaras, posicionadas em três distintos ângulos de visão – frente, lateral esquerda e lateral direita. O objectivo inicial era obter uma visão geral de 180º de toda a frente do robô, permitindo visualizar uma maior área de trabalho. Depois de adquirida a imagem de apenas uma câmara, verificou-se que o uso de mais câmaras era desnecessário, pois a informação obtida com apenas uma câmara já permite detectar bolas de golfe com um ângulo máximo de 70º, verificando-se ser suficiente para o correcto controlo e detecção das bolas de golfe.

Testaram-se duas resoluções distintas de aquisição de imagem:

- Resolução 320x240 e
- Resolução 640x480

Inicialmente fez-se aquisição de imagem na resolução 320x240, verificando-se que esta resolução não era adequada para a detecção correcta de bolas de golfe. Com aquisição de imagem nesta resolução é necessário um ajuste dos parâmetros de detecção de bolas (raio, tolerância de cor e tolerância de factor de forma circular) com valores muito baixos, implicando uma maior taxa de erro na detecção de bolas de golfe em pequenas variações de luminosidade devido à escassa informação que define o contorno redondo da bola.

Através da aquisição à resolução de aquisição máxima permitida pela câmara (640x480) consegue-se uma melhor calibração dos parâmetros de detecção de bolas. Depois de ajustados os parâmetros à nova resolução, constataram-se outros problemas, são eles:

- Má adaptação dos sistemas de ajuste automático (AGC, BLC, etc.), ao ambiente exterior com incidência de sol em variadas alturas do dia – necessário ajuste manual para diferentes horas de funcionamento.
- Problemas de reflexos do sol na relva – originam aquisição com elevada saturação da imagem e consequente detecção errada de bolas.

A *Figura 30* mostra alguns dos problemas encontrados.

Figura 30 *Detecção errada de bolas de golfe devido ao mau funcionamento dos sistemas de ajuste automático AGC e BLC e reflexos da relva.*

Devido aos reflexos da luz solar no relvado, os sistemas de ajuste automático não compensam a elevada quantidade de luz reflectida, originando uma aquisição de imagem com tons de cor muito acentuados.

De modo a diminuir os efeitos deste problema fez-se um ajuste manual nos parâmetros da câmara de vídeo analógica nomeadamente, no nível DC de ajuste do sinal de saída. Obtendo como resultado final uma imagem mais escura, apresentada na *Figura 31*.

Figura 31 *Resultado do processamento com ajuste manual do nível DC da câmara*

Este tipo de ajuste apresenta a desvantagem de não ser fiável na implementação de sistemas autónomos, pois é necessária uma calibração manual sempre que varia a iluminação exterior.

Com compensação por software não se conseguiram obter resultados satisfatórios devido à perda de informação no processo de aquisição. A solução ideal será a aquisição de uma nova câmara de vídeo, que apresente melhor qualidade de cor de modo a adquirir correctamente a imagem do campo, diminuindo o efeito dos reflexos da luz solar na relva.

4.2 Filtros na imagem

A *Figura 32* apresenta o resultado do processamento de imagem com resolução 320*240 aplicando diferentes filtros passa-baixo com realce de contornos através da função `cvCanny( B, B, 0, 100, 3)`.

Conseguem-se visualizar as diferenças entre a aplicação de filtros passa-baixo 3x3 e filtros passa-baixo 9x9, bem como as diferenças entre filtros de suavização por

aproximação “simple blur” e filtros de suavização complexos como é o caso do filtro implementado nas funções do OpenCV “gaussian blur”.

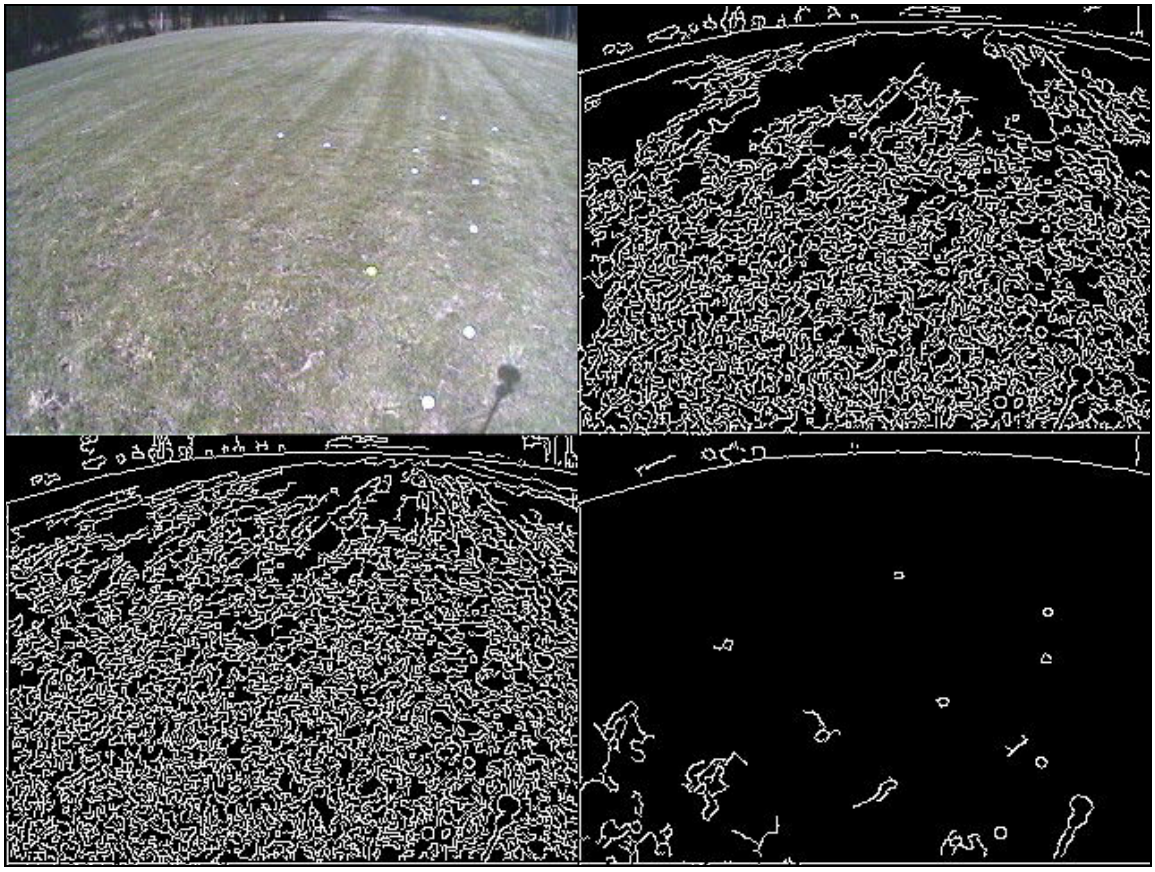


Figura 32 *Imagem original com aquisição 320x240 e detecção de contornos com filtragem a)"simple blur" 3x3 b)”gaussian blur” 3x3 c)”gaussian blur” 9x9, respectivamente.*

Através da filtragem baseada em máscaras `nxn` escaladas 1/`nxn` não é possível detectar os contornos de bolas de golfe devido ao elevado ruído presente na imagem.

O filtro que melhor se adapta à detecção de contornos de bolas, com resolução de imagem 320*240 e realce de contornos com parâmetros fixos, é o filtro “gaussian blur” 9x9 aplicado através da função OpenCV `cvSmooth(B, B, CV_GAUSSIAN, 9, 9, 0, 0)`.

A maior desvantagem na utilização do filtro de suavização aplicando a distribuição de gauss é o elevado tempo de processamento. No final de alguns ajustes constatou-se que haviam diferenças significativas ao variar os parâmetros do algoritmo de realce de contornos.

A *Figura 33* mostra o resultado do processamento de imagem a 640x480 com filtro de suavização "simple blur" 3x3 e detecção de contornos com parâmetros do algoritmo de Canny thresh1=325, thresh2=400 e aperture=3 (`cvCanny( B, B, 325, 400, 3)`).

Figura 33 *Imagem com resultado final do processamento com aplicação do filtro "simple blur" 3x3 e detecção de contornos com parâmetros 325, 400,3 – resolução 640x480.*

O resultado obtido com a variação dos parâmetros de realce dos contornos é satisfatório tanto a nível de tempo de processamento (conforme referido em 4.4), como a nível de detecção correcta de bolas – com os parâmetros actuais (4<raio<9, cor_centro>250, tolerância_circulo=20) consegue-se detectar a maior parte das bolas, havendo apenas o problema de detecção de bolas nos extremos da imagem e a não detecção de bolas com contornos juntos devido à limitação de raio.

4.3 Utilização de histograma de cores

O uso de histogramas de cor para fazer compensação por software de variações de luminosidade para diferentes níveis de incidência solar no campo de golfe, verificou-se pouco satisfatória. Isto devido ao facto da imagem adquirida pela placa de aquisição nos casos de muita luminosidade perder informação relevante – a imagem adquirida tem uma forte incidência de brancos conforme descrito na *Figura 30*.

Através do uso do histograma efectua-se um ajuste dos valores de cor de todos os pixeis da região de processamento, originando uma imagem mais escura e não se obtém maior realce das zonas de contorno das bolas.

Efectuou-se o teste do histograma de cor através do seguinte código:

```
CvHistogram *hist; //estrutura que guarda os dados do histograma
int hdims=256; //diferentes cores a representar no histograma
```

```
…
     //inciar a estrutura do histograma
     hist = cvCreateHist( 1, &hdims, CV_HIST_ARRAY, ranges, 1);
     //calcular o histograma da imagem de cinzentos
     cvCalcHist( &src, hist, 0, NULL );
     limiar=otsu(hist); //calcular limiar de threshold
     //mostrar histograma
     ShowHistogram("Histograma cinzentos", hist, limiar);
```

Devido à opção de detecção de bolas através do seu contorno, a aplicação do histograma não aumenta a capacidade de detecção correcta de bolas de golfe, por isso optou-se por não implementar uma solução com uso de histograma diminuindo assim processamento repetitivo.

4.4 Comparação de tempos de processamento

De modo a obter uma relação de comparação relativa à limitação de tempo imposta pelo processamento de imagem, fez-se a medição de tempos de processamento utilizando as seguintes linhas de código:

```
struct timeval tv;
double time;                      //variavel tempo de processamento
…
gettimeofday(&tv, NULL);          //ler tempo antes do processamento
time=tv.tv_sec+tv.tv_usec*1e-6;   //guardar tempo em segundos

rotina_de_processamento();        //funcao a medir tempo de processamento

gettimeofday(&tv, NULL);          //ler tempo apos processamento
time=(tv.tv_sec+tv.tv_usec*1e-6) - time; //guardar tempo total
```

Antes da chamada da rotina de processamento de imagem faz-se a leitura do tempo do sistema em milisegundos, seguindo-se a rotina de processamento e uma nova leitura do tempo do sistema. O tempo total de processamento é a diferença entre as duas medições.

Para testar a performance do sistema de processamento de imagem compararam-se os tempos de processamento com o uso de diferentes técnicas de filtragem e realce de contornos. Entre as várias técnicas possíveis, compararam-se duas das que apresentaram melhores resultados de processamento e correcta detecção de bolas – variando os parâmetros do filtro de suavização e os parâmetros do filtro de realce de contornos.

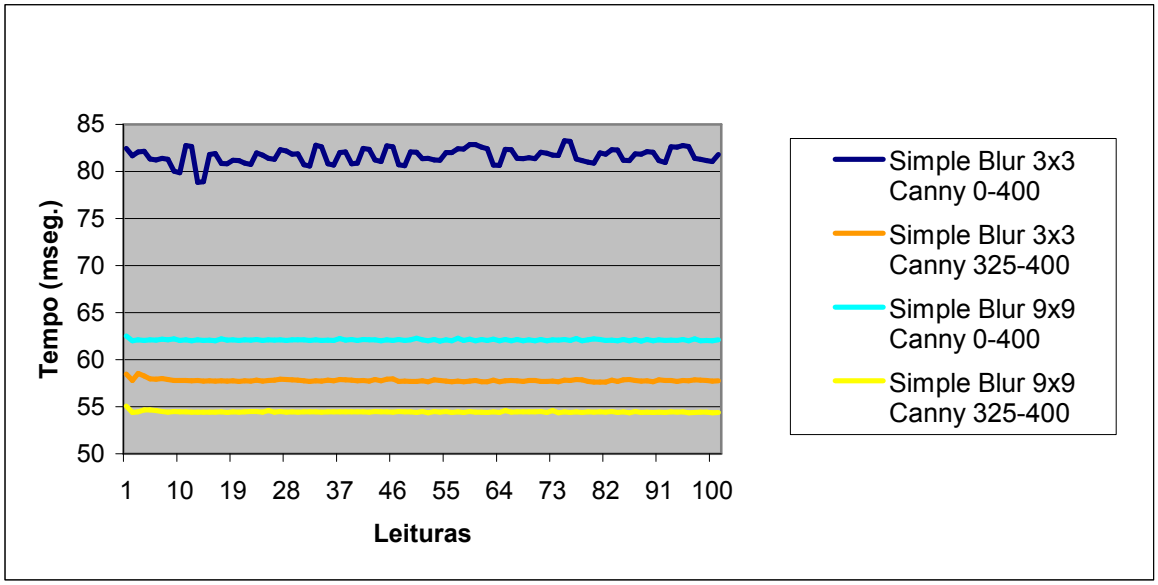


Figura 34 *Tempo de processamento de imagem com filtro simple blur*

O filtro de suavização que apresenta melhor resultado de processamento é o filtro "simple blur" 3x3 e realce de contornos com parâmetros de filtro de Canny 325-400. Através da *Figura 34* verifica-se que o uso deste filtro faz com que existam baixos tempos de processamento – entre 55 e 60 mseg, que equivalem a 16.7 e 18.2 fps respectivamente. Aumentando a máscara de processamento para 9x9 consegue-se menor tempo de processamento mas com um pior resultado de detecção de bolas.

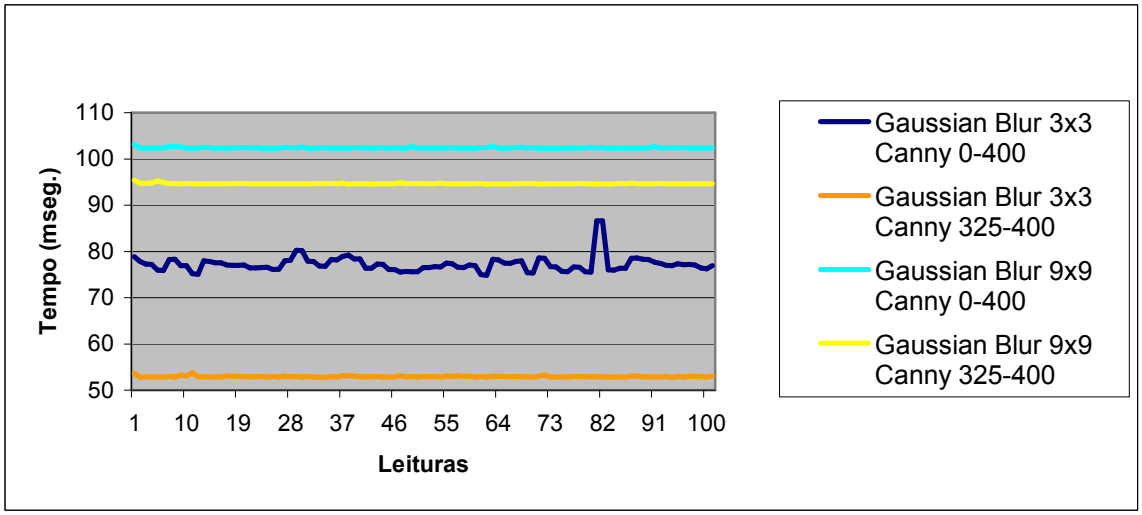


Figura 35 *Tempo de processamento de imagem com filtro gaussian blur*

A *Figura 35* mostra os tempos de processamento para o filtro de suavização por distribuição de Gauss.

Através das figuras *Figura 34* e *Figura 35* verifica-se a existência de tempos de processamento algo instáveis e com tempo médio de 80 mseg, relativos ao uso do realce

de contornos com parâmetros 0-400, donde se pode concluir que o algoritmo de Canny apresenta processamento rápido para parâmetros de processamento com distância entre eles idêntica aos usados nos parâmetros 325-400.

4.5 Adaptação do conhecimento base

Após a detecção dos contornos na imagem, foi necessário estabelecer regras que definem quais os contornos correspondentes ao objecto que se pretende isolar. Para isso foram efectuadas diversas tentativas de detecção e ajuste das formas circulares.

As variáveis que sofreram alguns ajustes foram:

- Gama de raios que definem uma bola de golfe na imagem – necessária para desprezar possível ruído inerente da fase de filtragem.
- Limites laterais da imagem sobre os quais não devem ser detectadas bolas de golfe – necessário para a não detecção de meios contornos que podem ser originários de formas não circulares.
- Valor mínimo da cor no centro de uma bola – embora a bola seja por defeito branca (255,255,255), pequenas variações de iluminação podem originar diferentes tons de cor.
- Factor de forma circular e tolerâncias – a detecção de contornos e o cálculo das características dos mesmos (área e perímetro), têm tolerâncias que se devem ter em atenção.

Estes parâmetros estão definidos na função de detecção de contornos `cvContours()` descrita em 3.7.

4.6 Transmissão de dados e imagem

A transmissão de dados é uma parte fundamental do sistema de detecção de bolas e transmissão da informação para a unidade de processamento contínuo. Através do valor de direcção encontrado após o processamento de imagem, a FoxBoard controla os motores de permitindo deslocar o robô com a direcção ideal para uma correcta recolha das bolas detectadas.

Inicialmente efectuou-se a transmissão de dados e imagem usando o protocolo de transporte TCP. Verificando-se um funcionamento incorrecto e atrasos de recepção de

imagem que impõe o não sincronismo entre *frames* recebidas – quando há um erro na comunicação sem fios, devido ao aumento da distância de comunicação, são recebidas frames repetidas e/ou dessincronizadas.

Depois de adoptada nova topologia de transmissão de dados, constatou-se que a transmissão de imagem por TCP não é o ideal, devido aos constantes bloqueios na transmissão sempre que existe uma perda de trama de imagem, necessitando de um novo processo de conexão.

A transmissão de dados por comunicação sem fios (*wireless)* tem a desvantagem de apresentar um aumento da taxa de erros de transmissão quando aumenta a distância entre dispositivos de comunicação. Por isso optou-se por implementar um sistema de transmissão de dados baseado num protocolo de comunicação sem conexão – UDP. Este protocolo permite o constante envio de dados sem qualquer controlo de erros, que no caso do envio de imagem não é um problema crítico e torna-se uma vantagem devido ao objectivo de visualização remota dos "olhos" do robô em sincronia com o mesmo. Quem visualiza a interface gráfica no pcRemoto não necessita de visualizar imagens perdidas mas sim sequência de imagens actualizadas e sincronizadas.

Na transmissão de dados os resultados com implementação do protocolo UDP são aceitáveis, apenas se verificando por vezes a não activação de pedidos com o accionamento único da tecla respectiva, sendo necessário efectuar o pedido duas ou mais vezes simultâneas, devido ao facto do protocolo UDP não implementar sistemas de controlo de fluxo e garantia de envio de dados. No entanto, como se verificou a correcta recepção dos pedidos fulcrais ao bom funcionamento do sistema, optou-se por não adoptar protocolos de comunicação distintos para a comunicação de dados e de imagem.

Depois de implementadas as rotinas de comunicação de imagem no pcLocal e no pcRemoto procedeu-se ao registo dos tempos de aquisição remota de imagem por *wireless*, com o servidor distanciado do cliente em três posições distintas: a menos de 5 metros, a 25 metros e a 50 metros do cliente. As três distintas posições permitem comparar as diferenças de taxa de transmissão com o aumento da distância entre dispositivos de comunicação.

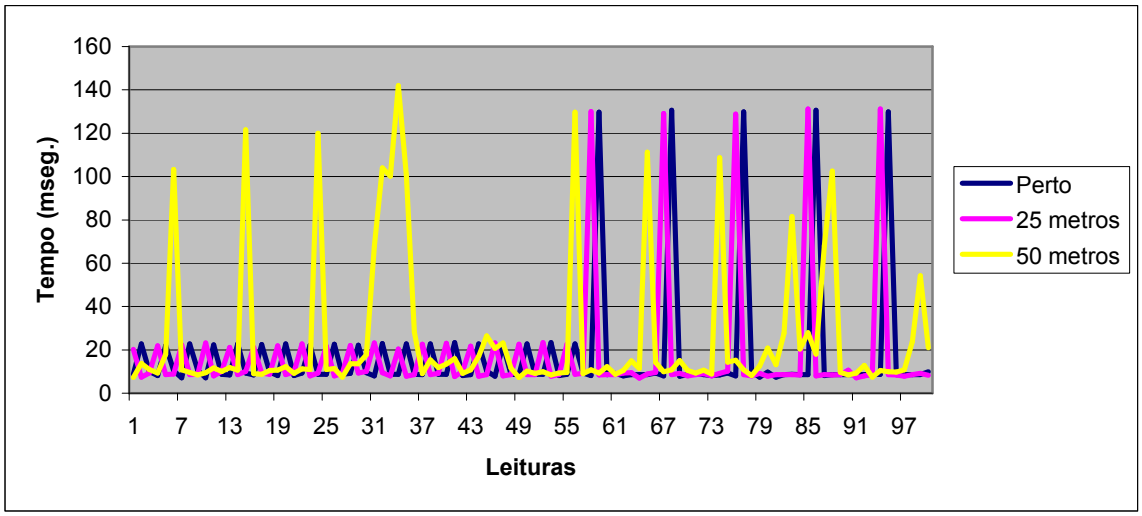


Figura 36 *Tempos de recepção de imagem a diferentes distâncias pcLocal-pcRemoto*

Na *Figura 36* apresentam-se os tempos de recepção de imagem no pcRemoto. Neste gráfico, verifica-se a diferença entre os tempos de recepção de imagem sem processamento de imagem a correr no pcLocal (leituras 1 a 57) e os tempos de recepção de imagem com processamento (leituras 58 a 100).

Ao activar o processamento de imagem (`flag_processar=1`) os tempos de recepção são acrescentados aos tempos de processamento de imagem do pcLocal relatados em 4.4 com o filtro "simple blur 3x3, Canny 325-400".

De salientar a diferença entre os tempos de recepção de imagem para distâncias menores que 25 metros e distâncias próximas dos 50 metros. No gráfico da *Figura 36* nota-se uma recepção com picos de tempos entre os 100 e os 140 mseg, com e sem processamento de imagem no pcLocal. Isto deve-se à ampliação do ruído inerente da linha de transmissão sem fios, pois para distâncias próximas dos 50 metros o receptor *wireless* apresenta uma recepção de sinal fraca. A solução para o aumento da distância de transmissão é a aplicação de antenas de ampliação do sinal a transmitir e o uso de receptores sem fios com maior gama de alcance, obtendo assim uma transmissão de imagem mais aproximada da transmissão em tempo real.

Capítulo V Discussão

Porque se utilizam câmaras, vantagens, problemas e soluções?

A ideia de utilização de câmaras no robô advém da necessidade de detectar objectos com cor forte num fundo distinto – bolas de golfe brancas num fundo verde (relvado). Esta opção verificou-se fiável, obtendo-se um resultado de detecção de bolas que permite a correcta recolha das mesmas.

Ao utilizar a visão por computador podem-se implementar diferentes soluções não limitadas apenas à detecção de bolas de golfe, como é exemplo a detecção de objectos no campo de treinos – normalmente o campo *driving range* tem ao longo do seu comprimento barreiras indicadores da distância à zona de batimentos, pelo que estas podem ser um possível problema na deslocação da plataforma. Definindo-se uma cor padrão para essas barreiras e usando um sistema visão local é possível detectar e definir as barreiras como sendo obstáculos, permitindo uma correcta e atempada correcção de rota de recolha das bolas.

Uma das desvantagens da utilização de visão local no robô é a necessidade de se efectuar uma primeira procura ao longo de todo o campo de modo a obter informação das zonas de maior concentração de bolas onde é prioritária a recolha das mesmas. Com a implementação de um sistema de localização de bolas com auxílio de micro dispositivos inseridos no interior de cada bola no processo de criação das mesmas e adaptação dos dados a sistemas de posicionamento GPS, este problema poderia ser eliminado obtendo-se assim uma maior taxa de recolha de bolas por unidade de tempo e uma correcta detecção das bolas de golfe. Esta solução tem a principal vantagem da detecção de bolas mesmo sem ser em campos de treinos de golfe, podendo ser inseridos *microchips* em todo o tipo de bolas de golfe permitindo uma melhor procura das bolas no decorrer de um jogo normal de golfe. Como inconveniente principal existe a dificuldade de incluir o *microchip* nas bolas de golfe, com a garantia que este funcione correctamente durante o tempo de vida útil da bola e o elevado custo de produção necessário para uma fase de protótipo.

A necessidade de uma unidade de processamento extra é um incentivo para o uso de outros sistemas de correcta detecção de bolas de golfe. Caso se consigam obter modos de detecção de bolas golfe de fácil implementação e sem a utilização de câmaras localmente, a comunicação de todo o sistema reduz-se apenas à actualização de sensores no pcRemoto. Permitindo a diminuição, de toda a estrutura de protecção das unidades de processamento, para dimensões limitadas apenas pelo tamanho do hardware de controlo de motores, pela unidade de processamento contínuo e pelo dispositivo de comunicação sem fios.

Utilização de várias câmaras

A utilização de visão computacional, com aquisição proveniente de várias câmaras posicionadas localmente no robô apresenta um nível de complexidade de processamento elevado. É necessário impor uma sincronia entre todas as imagens adquiridas bem como tempos de processamento diminutos entre rotinas de processamento de imagem.

Para efectuar um processamento multi câmara será necessário um computador com elevada velocidade de processamento que permita o funcionamento em paralelo de várias rotinas de processamento de imagem que imponham o sincronismo entre câmaras e permitam o calculo correcto da direcção a transmitir à unidade de processamento contínuo.

Detecção de bolas por contornos e características padrão de círculos

A opção de uma metodologia de processamento de imagem baseada na detecção de contornos dos objectos seguida de comparação de cor do ponto central e aceitação do objecto quando este respeita valores de raio, área e perímetro, apresenta resultados satisfatórios. No entanto, existe ainda o problema da não detecção de bolas cujos contornos estão agrupados entre si. Quando se verificam bolas encostadas, implicando um contorno com maior área, perímetro e maior raio do círculo envolvente, o sistema não tem apresenta qualquer método de correcção do problema, desprezando o contorno. Este pode vir a ser um possível problema, se ao longo do tempo se verificar que existe uma tendência das bolas de golfe caírem em posições do campo que as façam deslizar de modo a juntarem-se. Actualmente não se verificaram muitos casos deste tipo mas, uma possível solução para dotar o sistema de um modo de resolução, pode passar por uma procura de contornos seguido de aplicação de algoritmos de acumulação de valores que

permitam a detecção de círculos respeitando a equação geral de circunferência – transformada de Hough para detecção de círculos.

A transformada de Hough aplica a equação geral da circunferência (eq. 2), percorrendo todos os pontos detectados pelo filtro de realce de contornos, seguindo-se uma acumulação dos valores padrão da equação da circunferência – raio e centro (x_i,y_i), para os diferentes pontos (x,y) da imagem.

O uso de diferentes esquemas representativos de cor. Porquê RGB e porque não HSV?

Segundo a pesquisa bibliográfica efectuada, é comum utilizar câmaras que adquirem imagem no formato standard RGBA, seguindo-se a conversão da imagem para espaços de cor alternativos, como é o caso do espaço de cor HSV. Com esta conversão é normal conseguir uma diminuição de taxa de ruído da imagem, bem como melhor detecção de cores.

No presente trabalho optou-se por efectuar um processamento de imagem directamente do formato RGB. Isto porque não existe a necessidade de detecção de cores específicas, apenas se necessita de realçar contornos com tempos de processamento adequados à velocidade máxima do robô.

Limite de distância máxima para monitorização de sensores.

Através do teste e medidas de tempos de transmissão de dados descritos em 4.6, verificou-se a necessidade de ampliação do sinal a transmitir para permitir uma melhor monitorização dos sensores e imagem. Este problema deve-se à necessidade de diminuir ao máximo o número de dispositivos dispostos no exterior do invólucro de protecção das unidades de processamento – o transmissor de sinal sem fios está disposto no seu interior para que não haja o risco de danificá-lo.

O teste de inclusão de pontos retransmissores de sinal ao longo do campo de treinos de golfe ou a utilização protecção para as unidades de processamento baseada em material não metálico são soluções que necessitam de serem testadas para que se possa concluir se existe ou não a necessidade de retirar o transmissor sem fios do interior do invólucro de protecção.

Capítulo VI Conclusões

A utilização da biblioteca OpenCV simplificou todo o processo de teste e implementação de algoritmos de detecção de bolas de golfe, pois não foi necessário adquirir conhecimentos muito aprofundados de como tratar digitalmente uma imagem. Através desta biblioteca conseguiram-se descartar em pouco tempo de implementação, soluções que inicialmente pareciam ideais e de maior simplicidade e que se verificaram não fiáveis para o problema a resolver, como é exemplo o uso de equalização de histogramas para compensar variações de iluminação.

O uso de visão por computador para a detecção de objectos em ambientes exteriores mostrou ser um trabalho algo complexo. Os sistemas de detecção de defeitos ou detecção de objectos utilizados em ambientes industriais, contêm sistemas de controlo de iluminação adequada para o correcto realce das características do objecto a detectar. No caso particular deste trabalho não é realizável qualquer tipo de controlo de iluminação, devido ao constante deslocamento do robô e ao limite de energia fornecida pelas baterias de alimentação, havendo a necessidade de adaptar todo o sistema aos diferenciados tipos de incidência solar – calibração manual do nível DC da câmara e ajuste por software dos parâmetros de tolerância das características de detecção de bolas de golfe. A necessidade de calibração manual verifica-se devido a uma má escolha da câmara usada para o sistema de visão.

Os critérios escolhidos para compra da câmara de vídeo não foram os mais satisfatórios: escolheu-se como critério principal um correcto funcionamento em ambientes exteriores e boa relação sinal/ruído, descartando alguma qualidade de cor da imagem e sistemas de compensação de cor. Futuramente deverá se optar por uma câmara que permita o ajuste por software de diferentes parâmetros: regular o nível de compensação de cor, ajustar percentagem de cor da imagem que activa/desactiva o AGC, AWB, etc.

A solução implementada neste trabalho faz parte da patente portuguesa nº 103807, com registo efectuado em 13 de Agosto de 2007 com a designação *"Sistema de recolha*

de bolas de golfe totalmente autónomo ou remotamente operado", Na qual se descreve em pormenor a plataforma móvel autónoma de recolha de bolas de golfe.

Esta solução funciona de modo eficaz na resolução do problema a solucionar – auxílio de posicionamento da plataforma móvel de recolha de bolas de golfe, efectuando uma correcta detecção e transmissão da "direcção" onde se detecta maior concentração de bolas.

6.1 Principais dificuldades sentidas durante a realização do trabalho

As principais dificuldades sentidas baseiam-se maioritariamente na adaptação e calibração do sistema de visão. O processo de adaptação, da visão adquirida pela câmara analógica às bolas de golfe e relvado foi algo moroso, devido ao facto de se utilizar uma câmara que inicialmente parecia ter a adequada representação de cor para ambientes exteriores, o que se veio a verificar não ser o ideal em locais com forte incidência solar e constantes reflexos inerentes do relvado normal do campo de práticas de golfe.

Outra das dificuldades foi o conhecimento e aperfeiçoamento das técnicas de processamento digital de imagem. Todo o conteúdo aplicado neste trabalho foi fruto de constante pesquisa e experimentação das diferentes técnicas existentes no processamento digital de imagem através de OpenCV. Foi necessária a consulta de variada bibliografia referente aos diferentes assuntos e efectuar diversas comparações de métodos e adaptações dos mesmos aos objectivos a implementar.

Um processo não menos difícil e acima de tudo mais trabalhoso foi o processo de implementação da interface gráfica para monitorização de resultados obtidos. Despendeu-se muito tempo de projecto para algo que em principio seria de rápida implementação e que no final se mostrou algo trabalhoso, nomeadamente o desenho de gráficos em sistemas operativos "não gráficos" e a transmissão de dados por comunicação sem fios. Esta última devido a problemas inerentes da transmissão sem fios de longa distância, que origina maior taxa de erros de transmissão e consequente "ampliação" de problemas iniciais de implementação.

6.2 Trabalho futuro

- Detecção de bolas de golfe através do processamento da imagem adquirida do sistema de vídeo vigilância: esta é uma das opções possíveis para melhorar a detecção correcta e atempada de zonas de maior concentração de bolas de golfe sem a necessidade de gastos extra nem necessidade de uma primeira "ronda" do robô a todo o campo.
- Aplicação de algoritmos de controlo juntamente com o processamento de imagem: a detecção de bolas de golfe juntamente com um correcto controlo de informação redundante poderá optimizar todo o processo bem como aumentar a performance de todo o sistema relativo ao tempo de recolha da maior parte de bolas no campo de golfe.
- Criação de ambiente de visualização e controlo remoto através de página web: a monitorização de todo o sistema através de acesso por internet é uma opção a considerar em futuros melhoramentos, com vista a tornar o sistema adaptável a um utilizador sem conhecimentos aprofundados de programação.
- Armazenamento de dados de todos os sensores: permitirá dotar o sistema da percepção de trajectórias percorridas, possibilitando a correcção de trajectórias de modo a efectuar uma recolha de bolas de golfe melhorada.
- Auxilio por visão computacional na entrada para o porto de descarga de bolas e carregamento de baterias: para melhorar o processo de atracagem no porto de "repouso" do robô pode-se testar a implementação de algoritmos de procura de cor que auxiliem a entrada para descarga autónoma de bolas e recarga de baterias.

Referências

[1] STEVENS, W. Richard; FENNER, Bill; RUDOFF, Andrew M. – **Unix Network Programming: the sockets Networking API**, vol. I, 3rd ed, Addison-Wesley, 2004, ISBN 0-13-141155-1, pág. 95-98.

[2] STEVENS, W. Richard; FENNER, Bill; RUDOFF, Andrew M. – **Unix Network Programming: the sockets Networking API**, vol. I, 3rd ed, Addison-Wesley, 2004, ISBN 0-13-141155-1, pág. 239-241.

[3] GONZALEZ, Rafael C.; WOODS, Richard E. – **Digital Image Processing,** Prentice Hall, 2002, ISBN 0-201-18075-8, pág. 26-30.

[4] WANG, Yao; OSTERMANN, Jörn; ZHANG, Ya-Qin – **Vídeo Processing And Communications**, Prentice Hall, 2002, ISBN 0-13-017547-1, pág. .8-9,22.

[5] http://en.wikipedia.org/wiki/Golf#Hitting_a_golf_ball – **Noções básicas de golfe**, acedido em Junho de 2007.

[6] http://www.rbirangeballs.com/PhotoGallery.asp?ProductCode=BC%2FMHD – **Mini Hand Push Picker,** acedido em Junho de 2007.

[7] http://www.ubuntu.com/products/WhatIsUbuntu/desktopedition – **Ubuntu, desktop edition,** acedido em Agosto de 2007.

[8] http://www.svgalib.org/jay/beginners_guide/beginners_guide.html – **Svgalib tutorials**, acedido em Junho de 2007.

[9] http://opencvlibrary.sourceforge.net/CvReference – **OpenCV reference manual**, acedido em Outubro de 2007.

[10] http://www.belrobotics.com/robot/mower/products_ballpicker.html – **BelRobotics BallPicker**, acedido em Junho de 2007.

[11] http://en.wikipedia.org/wiki/Charge-coupled_device – **Charge-coupled device**, acedido em Agosto de 2007.

[12] http://www.elbex.co.jp/MemberFiles/0002/BLC.PDF – **Elbex, – BLC, Backlight Compensation**, acedido em Agosto de 2007.

[13] http://en.wikipedia.org/wiki/YCbCr – **Transformação de cor YCbCr,** acedido em Julho de 2007.

[14] http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MORSE/threshold.pdf – Morse, Bryan S. – **Lecture 4: Thresholding,** Brigham Young University, 2000, acedido em Julho de 2007.

[15] http://robotica.itam.mx/espanol/archivos/hsvspace.pdf – Darrin Cardani – **Adventures in HSV Space,** acedido em Julho de 2007.

[16] Canny, J.F. – **A computational approach to edge detection**, IEEE Trans. on Pattern Analysis and Machine Intelligence, November 1986.

[17] http://www.sciencedirect.com/science/article/B6V09-48TCV4N-5Y/2/91f551d124777f7a4cf7b18325235673 – YUEN, H.K.; PRINCETON, J.; ILLINGWORTH, J. and KITTLER, J. –**Comparative study of Hough Transform methods for circle finding.**

[18] FREEMAN, Herbert – **Computer processing of line-drawing images,** Computing Surveys, March 1974, pág. 57-97.

Bibliografia

[Gonza93] GONZALEZ, Rafael C.; WOODS, Richard E. – **Digital Image Processing,** Addison-Wesley, 1993, ISBN 0-201-50803-6.

[Marr82] MARR, David – **Vision: a computational investigation into the human representation and processing of visual information**, W. H. Freeman, 1982. ISBN 0-7167-1567-8.

[Davie90] DAVIES, E.R. – **Machine Vision: Theory, Algorithms, Practicalities**, Academic Press, 1990, ISBN 0-12-206090-3.

[Perei04] PEREIRA, Alexandre; POUPA, Carlos – **Como escrever uma tese: monografia ou livro científico - usando o Word,** 3ªed., Edições Sílabo, 2004, ISBN 972-618-350-2**.**

[Castl96] CASTLEMAN, Kenneth R. **– Digital Image Processing**, Prentice Hall, 1996, ISBN 0-13-211467-4.

[Sousa05] SOUSA, António Manuel Ribeiro de – **Dissertação de mestrado: Localização automática de objectos em sequências de Imagens,** Universidade do Minho, 2005.

[Queir01] QUEIROZ, José E. Rangel de; GOMES, Herman M. – **Introdução ao Processamento Digital de Imagens***,* Revista RITA, vol. VIII, 2001.

[Steve04] STEVENS, W. Richard; FENNER, Bill; RUDOFF, Andrew M. – **Unix Network Programming: the sockets Networking API**, vol. I, 3rded, Addison-Wesley, 2004, ISBN 0-13-141155-1.

[Rabba91] RABBANI, Majid; JONES, Paul W. – **Digital Image Compression Techniques**, SPIE, 1991, ISBN 0-8194-0648-1.

[Wang02] WANG, Yao; OSTERMANN, Jörn; ZHANG, Ya-Qin – **Vídeo Processing And Communications**, Prentice Hall, 2002, ISBN 0-13-017547-1.

[Fpgpt07] http://www.fpg.pt/gdc_admin/upload/id2/PDF/regras.pdf
Federação Portuguesa de golfe **– Regras de golfe,** acedido em Julho de 2007.

[Bicas07] http://www.scielo.br/pdf/abo/v65n5/a01v65n5.pdf
BICAS, Harley E. A; ÀVILA, Marcos P. – **Visão artificial,** acedido em

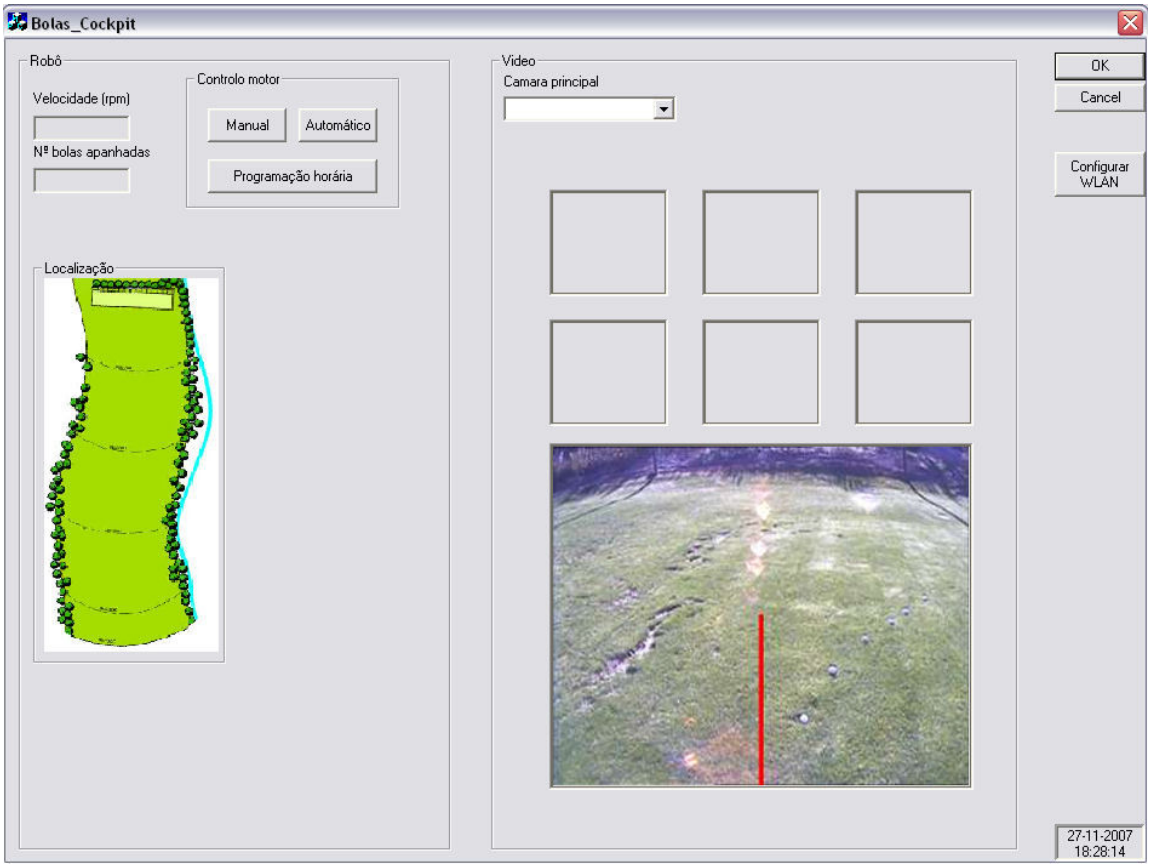
Junho de 2007.

[Thoma07] Thomas, Keir – **Beginning Ubuntu Linux 2ed**, APress 2007**,** ISBN 978-1-59059-820-7.

[Golfa07] http://www.golfama.com/iframegloss1.htm
Golfama, Definições básicas de golfe, acedido em Outubro de 2007.

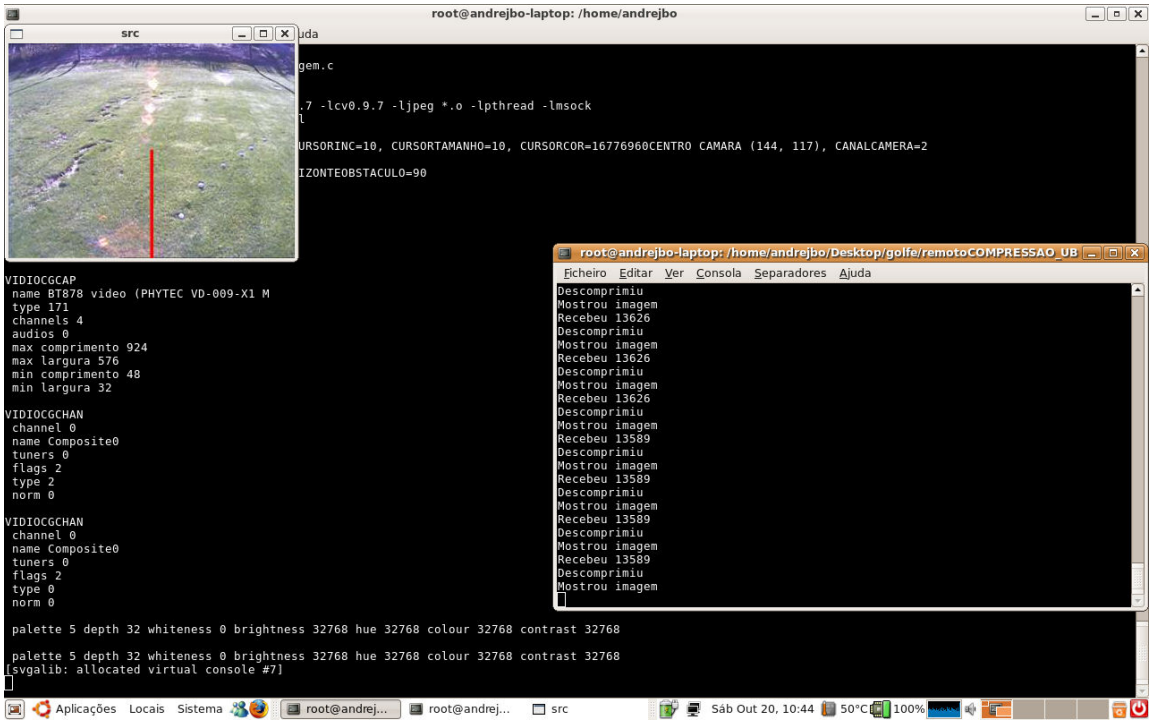
Anexos

Anexo I - Programa teste a executar em Windows



Programa de teste inicial, em ambiente Windows – programação Visual C++ MFC.

Anexo II - Programa teste a executar no linux



O *print-screen* acima mostra o resultado do programa de testes com a recepção de imagem de resolução 320x240 com compressão JPEG.

Anexo III - Código fonte do pcLocal

Local.c

```
1   ////////////////////////////////////////
2   //GOLFiNHO 2/11/2007
3   //Aplicação para processamento de imagem
4   ////////////////////////////////////////
5
6   #include <opencv/cv.h>
7   #include <opencv/highgui.h>
8
9   #include <vga.h>
10  #include <stdlib.h>
11  #include <stdio.h>
12  #include <pthread.h>
13  #include <math.h>
14  #include <string.h>
15  #include <linux/kd.h>
16  #include <sys/ioctl.h>
17
18  #include "videodev.h"
19  #include "geral.h"
20  #include "video.h"
21  #include "font.h"
22  #include "geometrias.h"
23  #include "rede.h"
24  #include "sensores.h"
25  #include "imagem.h"
26
27  int flag_gravarImagem=0,
28  flag_mostrarContornos=0,
29  flag_processar=1;
30
31  // Devolve 1 se numero par
32  //         0 caso contrário
33  int par(int num)
34  {
35    return (!(num%2));
36  }
37
38  static void restoretextmode(void) //sair do modo grafico
39  {
40    vga_setmode(TEXT);
41    vga_screenon();
42    ioctl( 1, KDSETMODE, KD_TEXT);
43  }
44
45  void imprimeDados(void)
46  {
47    vgaprintf( widthvideo/2, (heightvideo/2)-90, corCINZA,
48               "Inclin. PAN: %.2f TILT: %.2f  ",    T.pan, T.tilt);
49    vgaprintf( widthvideo/2, (heightvideo/2)-60, corCINZA,
50               "tempHumi: %.2f  ",                  T.tempHumi);
51    vgaprintf( widthvideo/2, (heightvideo/2)-50, corCINZA,
52               "Velocidade: %.2f  ",                T.velocidade );
53    vgaprintf( widthvideo/2, (heightvideo/2)-30, corCINZA,
54               "Humidade: %.2f  ",                  T.humidade);
55    vgaprintf( widthvideo/2, (heightvideo/2)-20, corCINZA,
56               "Bussula: %d  ",              T.bussula);
```

```
57    vgaprintf( widthvideo/2, (heightvideo/2)-100,corCINZA,
58               "GPS: Long %.2lf Lat %.2f ", T.longitude,T.latitude);
59    vgaprintf( widthvideo/2, (heightvideo/2)-80, corCINZA,
60               "Numero Sat: %d  ",                  T.nSat);
61    vgaprintf( widthvideo/2, (heightvideo/2)-70, corCINZA,
62               "tempGPS: %.2f  ",             T.tempGps);
63    vgaprintf( widthvideo/2, (heightvideo/2)-40, corCINZA,
64               "erro(%.2f,%.2f,%.2f) ", T.errox,T.erroy,T.erroz);
65    vgaprintf( widthvideo/2, (heightvideo/2)-10, corCINZA,
66               "velGPS: %.2f  ",              T.velGps);
67  }
68
69  void menuOpcoes(void)
70  {
71    char tecla=0;
72
73    tecla=vga_getkey();
74
75    if (tecla)
76    {
77      switch (tecla)
78      {
79        //captura imagem das !=s camaras --> 1= >fps
80        case '1':      flag_captura=((~(flag_captura&0x01))&0x01) |
81        (flag_captura&0x06) ;
82        break;
83        case '2':      flag_captura=((~(flag_captura&0x02))&0x02) |
84        (flag_captura&0x05) ;
85        break;
86        case '3':      flag_captura=((~(flag_captura&0x04))&0x04) |
87        (flag_captura&0x03) ;
88        break;
89        case 'c':      flag_mostrarContornos=!flag_mostrarContornos;
90        break;
91        case 'g':      flag_gravarImagem=!flag_gravarImagem;
92        break;
93        case 'q':
94        case 'Q':      flagThreadServidor=0;
95        break; // Termina o programa normalmente
96        case 'p':      flag_processar=!flag_processar;
97        break;
98
99      }
100     Velocimetro();
101   }
102   CBussola();
103 }
104
105 void funcao_imagem(void) //envio de imagem para o pcRemoto
106 {
107   //variaveis para compressao de imagem
108   unsigned char shrink[MAX_TAM_JPEG];
109   int tamanho_mem=0;
110   ////////////////////////////////////
111
112   init_envia_imagem(); //abre porta de comunicacao
113   while(1)
114   {
115     if(flagEnviarImagem) //envia sempre imagem ate novo pedido
116     {
117       //comprimir imagem para jpeg
```

```
118      tamanho_mem=Imagem_comprimir(ipl_shrink,shrink);
119
120      envia_imagem(shrink, tamanho_mem);
121     }
122   }
123   close_envia_imagem(); //nunca deve ca chegar
124  }
125
126  int main(int argc, char *argv[])
127  {
128   //variaveis auxiliares para guardar imagem
129   char nome_ficheiro[7];
130   int n_img=0;
131   ////////////////////////////////////////
132
133   int flag_video=0;
134   IplImage *ipl_img; //imagem original em estrutura opencv
135
136   // Leitura dos ficheiros de configuração
137   if ( !le_config_geral   (CONFIG_GERAL)          ||
138        !le_config_font          (CONFIG_FONT)     ||
139        !le_config_rede    (CONFIG_REDE)          ||
140        !video_init()
141        )
142         goto erro;
143
144   //iniciar biblioteca svgalib
145   vga_init();
146   vga_setmode(mymode);
147   ////////////////////////////
148
149   //placa de aquisicao de vídeo
150   if (vga_setlinearaddressing() == -1)
151   {
152    fprintf(stderr, "Could not set linear addressing.\n");
153    goto erro;
154   }
155
156   endereco_ecran = vga_getgraphmem();
157   endereco_video0= (map + gb_buffers.offsets[0]);
158   endereco_video1= (map + gb_buffers.offsets[1]);
159   endereco_video=endereco_video0;
160
161   if (DEBUG)
162   {
163    printf("endereco_ecran mapped at %08lx.\n",
164            (unsigned long) endereco_ecran);
165    printf("endereco_video[0] mapped at %08lx.\n",
166            (unsigned long) endereco_video0);
167    printf("endereco_video[1] mapped at %08lx.\n",
168            (unsigned long) endereco_video1);
169   }
170
171   if (!video_start(flag_video))
172    goto erro;
173   ////////////////////////////
174
175   //iniciar interface grafica
176   nome_programa(argv[0]);
177   mostraMapa();
178   mostra_robo(490,2);
```

```
179   init_sensores();
180   ////////////////////////////
181
182   //iniciar imagens do opencv
183   ipl_img=cvCreateImage(cvSize(widthvideo, heightvideo), 8, 4);
184   ipl_shrink=cvCreateImage(cvSize(widthvideo/2,heightvideo/2),8,4);
185   ////////////////////////////
186
187   // iniciar processamento paralelo para comunicação por rede
188   pthread_create(&thread_servidor, NULL, (void *)funcao_servidor,
189                  NULL); //inicio do servidor de dados
190   pthread_create(&thread_imagem, NULL, (void *)funcao_imagem,
191                  NULL); //inicio do servidor de imagem
192   ////////////////////////////////////////////////////////
193
194   do
195   {
196     //aquisicao da imagem
197     if (!captura_video(flag_video))
198       goto erro;
199
200     //converter para estrutura do opencv
201     memcpy(ipl_img->imageData, endereco_video, 640*480*4);
202
203     //Gravar imagem original
204     if (flag_gravarImagem)
205     {
206       sprintf(nome_ficheiro, "img%d.bmp",n_img++);
207
208       if(par(n_img))
209         cvSaveImage(nome_ficheiro, ipl_img); //gravar frame
210
211       if(n_img>200)
212         n_img=0;
213
214       flag_gravarImagem=0;
215     }
216
217     //processar imagem
218     if(flag_processar)
219       Processar(ipl_img);
220     else
221       //captar a 640*480 e reduzir para 320*240
222       cvResize( ipl_img, ipl_shrink, CV_INTER_LINEAR );
223
224     //apresentar imagem
225     ShowImage( ipl_shrink);
226     flag_video=!flag_video;
227
228     //inicio de envio das tramas
229     if (LIGAREDE)
230     {
231       if (flagChegouTrama)
232       {
233         //recebeu dados da FOX
234         imprimeDados();
235         actualiza_sensores();
236         flagChegouTrama=0;   //actualiza dados so uma vez
237
238         //sempre que chegam dados,enviar dados para o pcRemoto
239         flagEnviarDados=1;
```

```
240       }
241
242       if (flagEnviarDados)
243       {
244         // enviar trama de dados
245         enviar_trama((char *)&T, sizeof(TRAMADADOS),2);
246         flagEnviarDados=0; //enviar somente uma trama
247       }
248     }
249
250     //le opcao escolhida
251     menuOpcoes();
252
253     //calculo de fps
254     coracao();
255
256   } while (flagThreadServidor);
257   erro:
258     restoretextmode(); //sai do modo grafico do svgalib
259
260   //garantir fecho de comunicacao
261   enviar_trama( "VEL: 0 DIR: 0 ", 256, 0);
262   enviar_trama( "VEL: 0 DIR: 0 ", 256, 0);
263   enviar_trama( "VEL: 0 DIR: 0 ", 256, 0);
264   ////////////////////////////////
265
266   return EXIT_SUCCESS;
267 }
```

imagem.h

```
1   #include <opencv/cv.h>
2   #include <opencv/highgui.h>
3   #include <stdio.h>
4
5   #include "video.h"            //para endereco_video0
6   #include "geometrias.h"
7
8   extern int flag_mostrarContornos;
9   extern int flag_processar;
10
11  extern IplImage *ipl_shrink;
12
13  void mostra_robo();
14  void mostraMapa(void);
15
16  void Processar(IplImage *img); //a main do processamento de imagem
17  void img_gravarRaw(char *nome_fp, unsigned char* endereco,
18  int resolution); //gravar endereco em .raw
19
20  //definir contornos - bolas de golfe
21  void Contours(IplImage *img, IplImage *img_gray);
22  void ShowImage( IplImage *img); //mostrar IplImage no SvgaLib
23
24  //comprimir imagem para um tamanho máximo definido TAM_JPEG
25  int Imagem_comprimir(IplImage *orig, unsigned char *dest);
26  int Imagem_descomprimir(unsigned char *orig, IplImage *dest);
```

imagem.c

```
1   ////////////////////////////////////////////////////
2   //GOLFiNHO 2/11/2007
3   //funcoes de processamento de imagem e interface grafica
4   ////////////////////////////////////////////////////
5
6   #include "imagem.h"
7   #include "geral.h"
8
9   #include "font.h"
10  #include "rede.h"
11
12  #include "jmemio.h" //para compressao jpeg
13
14  #define centro_BussolaX 40
15  #define centro_BussolaY 440
16  #define raio_Bussola 20
17
18  IplImage *ipl_shrink; //imagem para interface grafica 320x240

19  void mostraMapa(void)
20  {
21  IplImage *mapa, *out;
22  int i;
23  int *endereco;
24  char* adr_CV;
25  int x_offset=0, y_offset=0;
26
27    //apresentar o .jpg do mapa do campo
28    mapa = cvLoadImage( "mapa.jpg" , 1 );
29    out = cvCreateImage( cvGetSize( mapa ), 8 , 4 );
30    cvCvtColor( mapa , out , CV_BGR2BGRA );
31
32    //mostra_imagem
33    x_offset=0;
34    y_offset=380-out->height;
35
36    adr_CV = out->imageData;
37
38    for( i=y_offset ; i<y_offset+out->height ; i++ )
39    {
40      endereco = ADDR_ECRAN( x_offset, i );
41      memcpy( endereco , adr_CV , (out->width)*4 );
42      adr_CV+=(out->width)*4;
43    }
44
45    //linha a volta da imagem do campo
46    rectangle(0,380-out->height,out->width,380,corG);
47
48    //libertar memoria
49    cvReleaseImage(&mapa);
50    cvReleaseImage(&out);
51  }
52
53  void mostra_robo(int x_offset, int y_offset)
54  {
55  IplImage *robo, *out;
56  int i;
57  int *endereco;
58  char* adr_CV;
59
60    //apresentar o .jpg do desenho do robô
```

```
61    robo = cvLoadImage( "robo.jpg" , 1 );
62    out = cvCreateImage( cvGetSize( robo ), 8 , 4 );
63    cvCvtColor( robo , out , CV_BGR2BGRA );
64
65    //mostra_imagem
66    adr_CV = out->imageData;
67
68    for( i=y_offset ; i<y_offset+out->height ; i++ )
69    {
70      endereco = ADDR_ECRAN( x_offset, i );
71      memcpy( endereco , adr_CV , (out->width)*4 );
72      adr_CV+=(out->width)*4;
73    }
74
75    //libertar memoria
76    cvReleaseImage(&robo);
77    cvReleaseImage(&out);
78  }
79
80  //guarda imagem sem qualquer compressao em formato .raw
81  void img_gravarRaw(char *nome_fp, unsigned char* endereco,
82  int resolution)
83  {
84  FILE *fp;
85
86    fp=fopen(nome_fp, "w+b");
87    fwrite(endereco, resolution, 1,fp);
88    fclose(fp);
89  }
90
91  void ShowImage( IplImage *img)
92  {
93    //mostrar imagem com janela de dimensoes padrão
94    copytoscreen_old(img->imageData,0,0);
95  }
96
97  //encontrar contornos e criar imagens com bolas de golfe detectadas
98  void Contours(IplImage *img, IplImage *img_gray)
99  {
100   CvSeq *contours, *contours_aux;
101   CvMemStorage *storage = cvCreateMemStorage(0);
102   CvFont font;
103   cvInitFont( &font, 0, 0.35, 0.35, 0, 1, 8);
104
105   float velocidade=0.0, direccao=0.0;
106   char str[100];
107
108   int countBolas=0;
109   CvPoint2D32f centroMassa;
110
111   contours = 0;
112   centroMassa.x=0; centroMassa.y=0;
113
114   IplImage *g=cvCreateImage( cvGetSize(img), 8, 1);
115   cvCopy(img_gray,g,NULL);
116   cvFindContours(g, storage, &contours, sizeof(CvContour),
117                  CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE,cvPoint(0,0));
118   cvReleaseImage(&g);
119
120   if(!contours)
121     return;
```

```
122
123   contours_aux = contours;
124
125   while(contours)
126   {
127   CvPoint2D32f centro;
128   float raio=0;
129   float perimetro=0.0;
130   float area=0.0;
131   float tolerancia=0.0;
132   float c=0.0;
133
134     cvMinEnclosingCircle( contours, &centro, &raio);
135     perimetro = cvArcLength( contours, CV_WHOLE_SEQ, -1 );
136     area = fabs(cvContourArea( contours, CV_WHOLE_SEQ));
137     c=(perimetro*perimetro)/area; //c=P*P/A=4*pi para o circulo
138     tolerancia=20;
139
140     //detectar bolas com gama de raios e gama de cores no centro
141     if(raio>=4 && raio<9
142       //testar se é um circulo
143       && c>=4*M_PI-tolerancia && c<=4*M_PI+tolerancia
144       //nao processar extremos da imagem
145       && centro.x>5 && centro.x<img->width-5
146       && centro.y>5 && centro.y<img->height-5
147
148       && (unsigned char) img->imageData[(int)centro.x*4+(int)centro.y
    *widthvideo*4]>254
149       && (unsigned char) img->imageData[(int)centro.x*4+(int)centro.y
    *widthvideo*4+1]>254
150       && (unsigned char) img->imageData[(int)centro.x*4+(int)centro.y
    *widthvideo*4+2]>254 )
151     {
152
153       sprintf(str, "Area= %.1f Perimetro= %.1f", area, perimetro);
154       vgaprintf(300,300,corG,str);
155
156       countBolas++;
157       centroMassa.x+=centro.x;
158       centroMassa.y+=centro.y;
159
160       //desenho de circlos detectados
161       cvCircle(img, cvPoint(centro.x, centro.y), raio,
162                cvScalar(255,0,0,0), 1, 8, 0);
163       cvCircle(img_gray, cvPoint(centro.x, centro.y), raio,
164                cvScalar(255,0,0,0), 1, 8, 0);
165     }
166     contours = contours->h_next;
167   }
168   contours = contours_aux;
169
170   //achar centro de Massa das bolas
171   centroMassa.x=centroMassa.x/countBolas;
172   centroMassa.y=centroMassa.y/countBolas;
173
174   if( countBolas)
175   {
176     if( centroMassa.x==widthvideo/2)
177     {
178       velocidade=120.0;
179       direccao=0.0;   // o centro da imagem e' direccao 0
```

```
180     }
181     else
182     {
183       velocidade=120.0;
184       direccao=centroMassa.x-widthvideo/2;
185
186       //converter para o maximo de direccao=vel/2
187       direccao=60*direccao/320;
188     }
189
190     direccao=-direccao;
191     sprintf(str,"VEL: %d DIR: %d ",(int)velocidade,(int)direccao);
192   }
193   else
194   {
195     velocidade=120; direccao=0;
196     centroMassa.x=320; centroMassa.y=240;
197
198     sprintf(str,"VEL: %d DIR: %d ",(int)velocidade,(int)direccao);
199   }
200
201   //enviar para FoxBoard direccao de centro massa das bolas de golfe
202   enviar_trama( str, 256, 0);
203
204   //indicar centro de massa com uma linha desde
205   //(widthvideo/2,heightvideo) até centro de massa
206   cvLine( img, cvPoint(img->width/2, img->height),
207           cvPoint(centroMassa.x, centroMassa.y),
208           cvScalar(0,0,255,0), 6,8,0);
209
210   //desenhar um circulo no local de centro de massa
211   cvCircle(img, cvPoint(centroMassa.x, centroMassa.y), 2,
212            cvScalar(0,0,255,0), 2, 8, 0);
213   cvCircle(img_gray, cvPoint(centroMassa.x, centroMassa.y), 2,
214            cvScalar(255,0,0,0), 2, 8, 0);
215
216   //indicar bolas detectadas e centro de massa
217   sprintf( str, "Num de Bolas: %d  CentroMassa: %.2f %.2f",
218            countBolas, centroMassa.x, centroMassa.y);
219   vgaprintf(5,240,corR,str);
220
221   //libertar memoria
222   cvReleaseMemStorage(&storage);
223 }
224
225 //função principal do processamento de imagem
226 void Processar(IplImage *img)
227 {
228   //imagem de 1 canal escala de cinzentos
229   IplImage *gray=cvCreateImage( cvGetSize(img), 8, 1);
230
231   //filtro passa-baixo "Simple-Blur" 3x3
232   cvSmooth(img, img, CV_BLUR, 3, 0, 0,0);
233
234   //converter BGRA para GRAY
235   cvCvtColor( img, gray, CV_BGRA2GRAY);
236
237   //realce de contornos
238   cvCanny( gray, gray, 325, 400, 3);
239
240   //detectar bolas de golfe
```

```
241   Contours( img, gray);
242
243   if(flag_mostrarContornos)
244   //se pedido de contornos, converter para 4canais de cor
245   //para enviar e apresentar no pcRemoto
246   cvCvtColor( gray,img, CV_GRAY2BGRA);
247   //captar a 640*480 e reduzir para 320*240
248   cvResize( img, ipl_shrink, CV_INTER_LINEAR );
249
250   //libertar memoria
251   cvReleaseImage(&gray);
252 }

253 /*****************COMPRESSAO DE IMAGEM**************************/
254 int Imagem_comprimir(IplImage *orig, unsigned char *dest)
255 {
256 unsigned long tamanho_jpeg=0;
257 IplImage *orig_;
258
259   orig_=cvCreateImage( cvGetSize(orig), 8, 3);
260   cvCvtColor( orig, orig_, CV_BGRA2RGB );
261
262   write_JPEG_memory ((unsigned char*)orig_->imageData,
263                      orig_->width, orig_->height, &dest[0],
264                      MAX_TAM_JPEG, QUALITY_JPEG, &tamanho_jpeg);
265
266   cvReleaseImage(&orig_);
267
268   return (int)tamanho_jpeg;
269 }
270
271 int Imagem_descomprimir(unsigned char *orig, IplImage *dest)
272 {
273 int w,h;
274
275   return (read_JPEG_memory ((unsigned char*)dest->imageData, &w,
276                              &h,&orig[0], MAX_TAM_JPEG));
277 }
```

rede.h

```
1   int le_config_IP(char *nomeficheiro);
2   int le_config_rede(char *nomeficheiro);
3
4   void enviar_trama(char msg[], int number_bytes, int ip);
5   void funcao_servidor(void);
6   void init_envia_imagem( void); // abrir porta de udp socket_imagem
7   void envia_imagem(unsigned char *endereco, int size);
8   void close_envia_imagem( void);
9
10  char IP_FOX[16];
11  char IP_PCREMOTO[16];
12  char IP_PCLOCAL[16];
13  extern unsigned short int PORT;
14  extern int LIGAREDE;
15  extern int socket_imagem;
16
17  extern char MEU_NUMERO_char;
18  extern int MEU_NUMERO_int;
19  extern int conta_TRAMAS_LIXO;
20
21  extern pthread_mutex_t mutex;
```

```
22  extern pthread_t thread_servidor;
23  extern pthread_t thread_imagem;
24  extern int flagThreadServidor, flagThreadImagem;
25
26  extern char BUFFER[MAXBUF];
27  extern char flagChegouTrama;
28  extern int flagEnviarImagem;
29  extern int flagEnviarDados;
```

rede.c

```
1   ////////////////////////////////////////////////
2   //GOLFiNHO 2/11/2007
3   //funcoes de transmissao de dados atraves de TCP/IP
4   ////////////////////////////////////////////////
5
6   #include <netinet/in.h>
7   #include <sys/socket.h>
8   #include <sys/ioctl.h>
9   #include <linux/kd.h>
10  #include <netdb.h>
11  #include <stdlib.h>
12  #include <stdio.h>
13  #include <string.h>
14  #include <unistd.h>
15  #include <string.h>
16  #include <errno.h>
17  #include <resolv.h>
18
19  // SERVIDOR
20  #include <arpa/inet.h>
21  #include <pthread.h>
22  #include <fcntl.h>
23
24  //Aquisicao remota de imagem
25  #include <msock.h>
26  #include <zlib.h>
27  #include <zconf.h>
28
29  #define CHECK_ERR(err, msg)
30  { \
31    if (err != Z_OK)
32    { \
33      fprintf(stderr, "%s error: %d\n", msg, err); \
34      exit(1); \
35    } \
36  }
37
38  #include "geral.h"
39  #include "font.h"
40  #include "video.h"
41  #include "rede.h"
42  #include "imagem.h"
43
44  //variaveis de processos paralelos
45  pthread_t thread_servidor;
46  pthread_t thread_imagem;
47  ////////////////////////////////
48
49  unsigned short int PORT=50000;
50  char BUFFER[MAXBUF];
51  char flagChegouTrama=1;
```

```
52  int flagThreadServidor=1;
53  int flagEnviarImagem=0, flagEnviarDados=0;
54
55  char IP_FOX[16];
56  char IP_PCREMOTO[16];
57  char IP_PCLOCAL[16];
58  int  LIGAREDE;
59  int  conta_TRAMAS_LIXO=0;
60  int socket_imagem;
61
62  //tramadados padrão
63  //assinatura[4], bateria12, bateria24, velocidade,gpsX, gpsY,
64  //xbeeX, xbeeY, pan, tilt, temperatura, humidade, bolas, bussula
65  TRAMADADOS T={ "GOLF",12,24,0.0,1,1,1,1,10.0,10.0,20,10.0,20,10};
66
67  int le_config_rede(char *nomeficheiro)
68  {
69  FILE *fp;
70  char caracter;
71  char str1[80], str2[128];
72
73    if ((fp=fopen(nomeficheiro, "r"))==NULL)
74    {
75      printf("Erro na leitura do ficheiro %s.\n", nomeficheiro);
76      return 0;
77    }
78
79    while (fscanf(fp,"%c",&caracter), !feof(fp))
80    {
81      if (caracter=='#')
82        fscanf(fp,"%*[^\n]%*c");
83        else
84      if (caracter!='\n')
85      {
86        str1[0]=caracter;
87        fscanf(fp, "%[^= ]=%[^\n]%*c", str1+1, str2);
88
89        //ler ip's dos PC's do sistema
90        if (!strcmp (str1, "FOX"))
91          sscanf(str2, "%[^ #]",IP_FOX);
92        if (!strcmp (str1, "PC_REMOTO"))
93          sscanf(str2, "%[^ #]",IP_PCREMOTO);
94        if (!strcmp (str1, "PC_LOCAL"))
95          sscanf(str2, "%[^ #]",IP_PCLOCAL);
96        if (!strcmp (str1, "PORT"))
97          sscanf(str2, "%hu", &PORT);
98        if (!strcmp (str1, "LIGAREDE"))
99          sscanf(str2, "%d", &LIGAREDE);
100     }
101   }
102   fclose(fp);
103   if (DEBUG)
104   {
105     printf("LIGAREDE=%d\n\n",LIGAREDE);
106     printf("FOX=%s\n",IP_FOX);
107     printf("PC_LOCAL=%s\n",IP_PCLOCAL);
108     printf("PC_REMOTO=%s\n",IP_PCREMOTO);
109     printf("PORT=%d\n",PORT);
110   }
111
112   return 1;
```

```
113 }
114
115 void enviar_trama(char msg[MAXBUF], int number_bytes, int ip)
116 {
117 int sock=0;
118 int length;
119 struct sockaddr_in server;
120 struct hostent *hp=0;
121
122 // SOCK_STREAM=TCP, SOCK_DGRAM=UDP
123 sock = socket(AF_INET, SOCK_DGRAM, 0);
124
125   if (sock < 0)
126   {
127    printf("\nErro socket");
128    return;
129    }
130
131    server.sin_family = AF_INET;
132
133    switch (ip)
134    {
135     case 0: hp = gethostbyname(IP_FOX); break;
136     case 1: hp = gethostbyname(IP_PCLOCAL); break;
137     case 2: hp = gethostbyname(IP_PCREMOTO); break;
138    }
139
140    bcopy((char *)hp->h_addr, (char *)&server.sin_addr,
141          hp->h_length);
142    server.sin_port = htons(PORT);
143    length=sizeof(struct sockaddr_in);
144
145    if ( sendto(sock, msg, number_bytes, 0,
146         (struct sockaddr*)&server, length) < 0 )
147    {
148     perror("Sendto");
149     printf(" %d Falhou\n",ip);
150    }
151
152   close(sock);
153 }
154
155 void funcao_servidor(void)
156 {
157 int sock=0;
158 int length, fromlen, bytes_recv;
159 struct sockaddr_in server;
160 struct sockaddr_in from;
161
162   sock = socket(AF_INET, SOCK_DGRAM, 0);
163
164   if( sock<0)
165   {
166    printf("\nErro abrir Socket envio  ");
167    return;
168   }
169
170   length = sizeof(server);
171   bzero(&server, length);
172   server.sin_family = AF_INET;
173   server.sin_port = htons(PORT);
```

```
174
175   if( bind(sock, (struct sockaddr*)&server, length)<0)
176   {
177    printf("\nErro binding  ");
178    return;
179   }
180
181   fromlen = sizeof(struct sockaddr_in);
182
183   while (flagThreadServidor)
184   {
185    bytes_recv = recvfrom( sock, BUFFER, MAXBUF, 0,
186                           (struct sockaddr*)&from,
187                           (socklen_t *) &fromlen);
188    if(bytes_recv<0)
189    {
190     printf("\nErro recvfrom");
191     return;
192    }
193    flagChegouTrama=0;
194
195    //tramas validas contem assinatura="GOLF"
196    if (strncmp(BUFFER, "GOLF", 4))
197    {
198     conta_TRAMAS_LIXO++;
199     flagChegouTrama=-1;
200    }
201    else
202    {
203     switch(BUFFER[4])
204     {
205      case 'I':
206       //sempre que recebe pedido de imagem inverte a flag
207         flagEnviarImagem=!flagEnviarImagem;
208       break;
209      case 'D':
210       //pedido de dados
211       flagEnviarDados=1;
212       break;
213      case 'C':
214       //pedido de contornos
215       flag_mostrarContornos=!flag_mostrarContornos;
216       break;
217      case 'P':
218       //pedido de processamento
219       flag_processar=!flag_processar;
220       break;
221      default:
222       //chegou trama da FoxBoard -->actualiza dados em memoria
223       flagChegouTrama=1;
224       sscanf(BUFFER, "%*[^,],%f,%f,%f,%f,%d,%f,%lf,%lf,%d,%f,%f,%f,%f,%f ",
225              &T.velocidade, &T.humidade, &T.pan, &T.tilt, &T.bussula,
226              &T.tempHumi,&T.longitude,&T.latitude,&T.nSat,&T.tempGps,
227              &T.errox,&T.erroy,&T.erroz,&T.velGps);
228       strcpy(BUFFER, " ");
229     }
230    }
231   }
232   close(sock);
233  }
234
```

```
235 #define SERV_PORT 51000 //porto padrao de transmissao de imagem
236
237 void init_envia_imagem( void)
238 {
239 struct sockaddr_in servaddr;
240
241   bzero(&servaddr, sizeof(servaddr));
242   servaddr.sin_family = AF_INET;
243   servaddr.sin_port = htons(SERV_PORT);
244   inet_pton(AF_INET, IP_PCREMOTO, &servaddr.sin_addr);
245
246   socket_imagem = socket(AF_INET, SOCK_DGRAM, 0);
247
248   connect(socket_imagem, (struct sockaddr *)&servaddr, sizeof(servaddr));
249 }
250
251 void envia_imagem(unsigned char *endereco, int size)
252 {
253   write(socket_imagem, endereco, size);
254 }
255
256 void close_envia_imagem( void)
257 {
258   close(socket_imagem);
259 }
```

Anexo IV - Código fonte do pcRemoto

Remoto.c

```
1   ////////////////////////////////////////////////
2   //GOLFiNHO 2/11/2007
3   //Aplicação para monitorizacao de sensores e imagem
4   ////////////////////////////////////////////////
5
6   #include <opencv/cv.h>
7   #include <opencv/highgui.h>
8
9   #include <vga.h>
10  #include <stdlib.h>
11  #include <stdio.h>
12  #include <pthread.h>
13  #include <math.h>
14  #include <string.h>
15  #include <unistd.h>
16  #include <linux/kd.h>
17  #include <sys/ioctl.h>
18  #include <vgakeyboard.h>
19
20  #include "videodev.h"
21  #include "geral.h"
22  #include "video.h"
23  #include "font.h"
24  #include "geometrias.h"
25  #include "rede.h"
26  #include "sensores.h"
27
28  #include "imagem.h"
29
30  int pedeVideo=0, pedeDados=0, ler_tempos=0; //variaveis para o
    menuOpcoes
31  char trama[100];
32  int prtscrn=0;
33
34  int X=0,Y=245; //posicionar o cursor
35
36  // Devolve 1 se numero par,
37  //         0 caso contrário
38  int par(int num)
39  {
40    return (!(num%2));
41  }
42
43  //sair do modo grafico quando bloqueia
44  static void restoretextmode(void)
45  {
46    vga_setmode(TEXT);
47    vga_screenon();
48    ioctl( 1, KDSETMODE, KD_TEXT);
49  }
50
51  void imprimeDados(void)
52  {
53    //converter para kms os valores de GPS
54    T.longitude=6378.0*cos(T.latitude*M_PI/180)*sin(T.longitude*M_PI/
      180);
55    T.latitude=6378.0*sin(T.latitude*M_PI/180.0);
```

```
56
57    //converter para metros os valores de GPS
58    T.longitude*=1000;
59    T.latitude*=1000;
60
61    vgaprintf(widthvideo/2, (heightvideo/2)-60, corCINZA,
62              "tempHumi: %.2f  ", T.tempHumi);
63    vgaprintf(widthvideo/2, (heightvideo/2)-50, corCINZA,
64              "Velocidade: %.2f  ", T.velocidade );
65    vgaprintf(widthvideo/2, (heightvideo/2)-30, corCINZA,
66              "Humidade: %.2f  ", T.humidade);
67    vgaprintf(widthvideo/2, (heightvideo/2)-20, corCINZA,
68              "Bussula: %d  ", T.bussula);
69    vgaprintf(widthvideo/2, (heightvideo/2)-80, corCINZA,
70              "Numero Sat: %d  ", T.nSat);
71    vgaprintf(widthvideo/2, (heightvideo/2)-70, corCINZA,
72              "tempGPS: %.2f  ", T.tempGps);
73    vgaprintf(widthvideo/2, (heightvideo/2)-40, corCINZA,
74              "erro(%.2f,%.2f,%.2f) ", T.errox,T.erroy,T.erroz);
75    vgaprintf(widthvideo/2, (heightvideo/2)-10, corCINZA,
76              "velGPS: %.2f  ", T.velGps);
77    vgaprintf(widthvideo/2, (heightvideo/2),corCINZA,
78              "GPS: Long %.2lf Lat %.2f ", T.longitude,T.latitude);
79  }
80
81  //comunicar a FoxBoard a posicao pretendida para o robo
82  void irPara(int x, int y)
83  {
84  float xx=0,yy=0;
85  char str[100];
86
87    xx=260*x/320; yy=60*y/124; //converterPosicao do ecran para
      coordenadas do campo
88
89    sprintf(str, "GOLFcoord%f,%f", xx, yy);
90    enviar_trama( str, strlen(str), 0);
91  }

92  char menuOpcoes(void)
93  {
94  char tecla=0;

95    tecla=vga_getkey();
96    fflush(stdin);
97    vgaprintf(440, 460, corW, "Tecla: %c", tecla); //mostra tecla
      seleccionada
98
99    switch (tecla)
100   {
101     //escolher imagem a receber
102     case '1':
103       pedeVideo=!pedeVideo; //variavel que  indica envio de qualquer
          imagem
104       pedeDados=0;
105       sprintf( trama, "GOLFI"); //Pedido de imagem
106       break;
107
108     //pedir actualizacao de dados - sensores, posicao, etc.
109     case '4':
110       pedeDados=1;
111       strcpy( trama, "GOLFD");  //Pedido de dados
```

```
112      break;
113    case 'e':  //parar receber imagem e dados
114      pedeVideo=0;
115      pedeDados=0;
116      break;
117    case 'c':
118      pedeDados=1;
119      strcpy( trama, "GOLFC"); //pedido de mostrar contornos
120      break;
121    case 'p':
122      pedeDados=1;
123      strcpy( trama, "GOLFP"); //pedido de mostrar processamento de
         imagem
124      break;
125    case 'l':
126      ler_tempos=!ler_tempos;
127      break;
128    case 'q':
129      flagSair=1;
130      break;
131    case 'D': X--; break; //teclas desenha cursor
132    case 'C': X++; break; //para definir posicao pretendida pro robo
133    case 'A': Y--; break;
134    case 'B': Y++; break;
135    case ' ': irPara(X,Y); //depois de posicionar o cursor...
       transmitir posicao pretendida
136
137    case 'P': prtscrn=1; break;
138   }
139   return tecla;
140 }
141
142 void funcao_imagem(void)
143 {
144 unsigned char img_comprimida[MAX_TAM_JPEG];
145 IplImage *ipl_img3ch, *ipl_img4ch;
146
147   //leitura de tempos de recepcao de imagem
148   struct timeval tv;
149   double time=0;

150   ipl_img3ch=cvCreateImage(cvSize(320, 240), 8, 3);
151   ipl_img4ch=cvCreateImage(cvSize(320, 240), 8, 4);
152
153   cvNamedWindow( "src",1);
154
155   init_recebe_imagem();
156
157   while(1)
158   {
159     time=0;
160     if(ler_tempos && pedeVideo) //se pretende fazer leitura de
       tempos de aquisicao de imagem
161     {
162       gettimeofday(&tv, NULL);
163       time=(tv.tv_sec)+(tv.tv_usec/1000000.0);
164     }
165
166     vgaprintf(440,439,corK,"                          ");
167     vgaprintf(440,430,corR,"Recebeu %d\n ",
                  recebe_imagem(img_comprimida, MAX_TAM_JPEG) );
```

```
168
169     //(src, dest) dest=imagem RGB de 3canais
170     Imagem_descomprimir(img_comprimida, ipl_img3ch);
171
172     cvCvtColor( ipl_img3ch, ipl_img4ch, CV_RGB2BGRA);
173
174     copytoscreen_old(ipl_img4ch->imageData, 0, 0); //mostra a imagem
        recebida
175
176     if(ler_tempos && pedeVideo)
177     {
178       gettimeofday(&tv, NULL);
179       time-=((tv.tv_sec)+(tv.tv_usec/1000000.0));
180       time=-time;
181       escreve_ficheiro("tempoRecvImgFnal.txt", time);
182     }
183   }
184
185   cvReleaseImage(&ipl_img3ch);
186   cvReleaseImage(&ipl_img4ch);
187
188   close_recebe_imagem();
189 }
190
191 int main(int argc, char *argv[])
192 {
193 char tc=0;
194
195   // Leitura dos ficheiros de configuração. Se algum falhar sai de
      imediato do programa
196   if ( !le_config_geral   (CONFIG_GERAL)          ||
197        !le_config_font          (CONFIG_FONT)     ||
198        !le_config_rede    (CONFIG_REDE)           )
199   {
200     printf("\nErro na leitura dos CONFIGs\n");
201     goto erro;
202   }
203
204   vga_init();       // Iniciar modo grafico SVGAlib
205   vga_setmode(mymode);    // Indica o mode grafico pretendido
206
207   if (vga_setlinearaddressing() == -1)
208   {
209     fprintf(stderr, "Could not set linear addressing.\n");
210     goto erro;
211   }
212
213   endereco_ecran = vga_getgraphmem();
214
215   if (DEBUG)
216   {
217     printf("endereco_ecran    mapped at %08lx.\n", (unsigned long)
        endereco_ecran);
218   }
219
220   pedeVideo=0;
221   pedeDados=0;
222
223   nome_programa(argv[0]);
224
225   X=20; Y=355;
```

```
226
227   mostra_imagem(0, 380-124, "mapa.jpg");
228   mostra_imagem(320, 2, "uminho.bmp");
229   mostra_imagem(640-128, 144, "gps.jpg");
230
231   init_sensores();  // desenha graficos dos sensores
232
233   // Ligas as threads Dados e Imagem
234   pthread_create(&thread_servidor, NULL, (void *)funcao_servidor, NULL);
      //fica em Listen no barramento TCP
235   pthread_create(&thread_imagem, NULL, (void *)funcao_imagem, NULL);
236
237   int pscr=0;
238   char pscr_[100];
239   do
240   {
241     desenha_cursor(X,Y,corG); //cursor do GPS
242
243     tc = menuOpcoes(); // Escolher receber imagem (1,2,3) ou dados e
      fazer o pedido a enviar.
244
245     if (pedeDados)
246     {
247       enviar_trama( trama, strlen(trama), 1);  //FOX=0 PCLOCAL=1
        PCREMOTO=2
248       //usleep(25000);
249       pedeDados=0;
250     }
251
252     if (pedeVideo )
253     {
254       enviar_trama( trama, strlen(trama), 1);  //FOX=0 PCLOCAL=1
        PCREMOTO=2
255       //usleep(25000);
256       pedeVideo=0;
257     }
258
259     if (flagChegouTrama) //se chegou trama de dados
260     {
261       imprimeDados();      //actualiza os dados
262       actualiza_sensores();
263       flagChegouTrama=0;
264     }
265
266     if(prtscrn==1) //fazer printscreen
267     {
268       IplImage *printScreen=cvCreateImage(cvSize(640, 480), 8, 4);
269
270       memcpy(printScreen->imageData, endereco_ecran, 640*480*4);
271       sprintf(pscr_, "printscreen%d.bmp",pscr);
272       cvSaveImage(pscr_, printScreen);
273       pscr++;
274       prtscrn=0;
275
276       cvReleaseImage(&printScreen);
277     }
278     coracao();
279
280   } while (!flagSair);
281   erro:
282     restoretextmode();
```

```
283
284   return EXIT_SUCCESS;
285   }
286
```

rede.h

```
1    int le_config_rede(char *nomeficheiro);
2
3    void enviar_trama(char msg[], int number_bytes, int ip);
4    void funcao_servidor(void);
5    void funcao_imagem(void);
6
7    char IP_FOX[16];
8    char IP_PCREMOTO[16];
9    char IP_PCLOCAL[16];
10   extern unsigned short int PORT;
11   extern int LIGAREDE;
12
13   extern int conta_TRAMAS_LIXO;
14
15   extern pthread_t thread_servidor;
16   extern pthread_t thread_imagem;
17   extern int flagThreadServidor, flagThreadImagem;
18
19   extern char BUFFER[MAXBUF];
20   extern char flagChegouTrama;
21   extern int flagEnviarDados;
22
23   // Rotinas para aquisicao de Imagem (pela rede)
24   void init_recebe_imagem( void);
25   int recebe_imagem(unsigned char *endereco, int size);
26   void close_recebe_imagem( void);
27
28   extern int socket_imagem;
```

rede.c

```
1    ////////////////////////////////////////////////
2    //GOLFiNHO 2/11/2007
3    //funcoes de transmissao de dados atraves de TCP/IP
4    ////////////////////////////////////////////////
5
6    #include <netinet/in.h>
7    #include <sys/socket.h>
8    #include <sys/ioctl.h>
9    #include <linux/kd.h>
10   #include <netdb.h>
11   #include <stdlib.h>
12   #include <stdio.h>
13   #include <string.h>
14   #include <unistd.h>
15   #include <string.h>
16   #include <errno.h>
17   #include <resolv.h>
18
19   // SERVIDOR
20   #include <arpa/inet.h>
21   #include <pthread.h>
22   #include <fcntl.h>
23
```

```
24  //Aquisicao remota de imagem
25  #include <msock.h>
26  #include <zlib.h>
27  #include <zconf.h>
28
29  #define CHECK_ERR(err, msg)
30  { \
31    if (err != Z_OK)
32    { \
33      fprintf(stderr, "%s error: %d\n", msg, err); \
34      exit(1); \
35    } \
36  }
37
38  #include "geral.h"
39  #include "font.h"
40  #include "video.h"
41  #include "rede.h"
42  #include "imagem.h"
43
44  //variaveis de processos paralelos
45  pthread_t thread_servidor;
46  pthread_t thread_imagem;
47  ////////////////////////////////
48
49  unsigned short int PORT=50000;
50  char BUFFER[MAXBUF];
51  char flagChegouTrama=0;
52  int flagThreadServidor=1;
53
54  int socket_imagem; // Esta variavel contem o socket do Servidor de
    Imagem
55
56  char IP_FOX[16];
57  char IP_PCREMOTO[16];
58  char IP_PCLOCAL[16];
59  int  LIGAREDE;
60  int  conta_TRAMAS_LIXO=0;
61
62  //tramadados padrão
63  //assinatura[4], bateria12, bateria24, velocidade,gpsX, gpsY,
    xbeeX, xbeeY, pan,   tilt,    temperatura, humidade, bolas, bussula
64  TRAMADADOS T={"GOLF",12,24,2.0,10,10,10,10,10.0,10.0,22,10.0,20,10};


65  int le_config_rede(char *nomeficheiro)
66  {
67  FILE *fp;
68  char caracter;
69  char str1[80], str2[128];

70    if ((fp=fopen(nomeficheiro, "r"))==NULL)
71    {
72      printf("Erro na leitura do ficheiro %s.\n", nomeficheiro);
73      return 0;
74    }
75
76    while (fscanf(fp,"%c",&caracter), !feof(fp))
77    {
78      if (caracter=='#')
79      fscanf(fp,"%*[^\n]%*c");
```

```
80      else
81      if (caracter!='\n')
82      {
83        str1[0]=caracter;
84        fscanf(fp, "%[^= ]=%[^\n]%*c", str1+1, str2);
85
86        if (!strcmp (str1, "FOX"))
87          sscanf(str2, "%[^ #]",IP_FOX);
88        if (!strcmp (str1, "PC_REMOTO"))
89          sscanf(str2, "%[^ #]",IP_PCREMOTO);
90        if (!strcmp (str1, "PC_LOCAL"))
91          sscanf(str2, "%[^ #]",IP_PCLOCAL);
92        if (!strcmp (str1, "PORT"))
93          sscanf(str2, "%hu", &PORT);
94        if (!strcmp (str1, "LIGAREDE"))
95          sscanf(str2, "%d", &LIGAREDE);
96      }
97    }
98    fclose(fp);
99
100   if (DEBUG)
101   {
102     printf("LIGAREDE=%d\n",LIGAREDE);
103     printf("FOX=%s\n",IP_FOX);
104     printf("PC_LOCAL=%s\n",IP_PCLOCAL);
105     printf("PC_REMOTO=%s\n",IP_PCREMOTO);
106     printf("PORT=%d\n",PORT);
107   }
108   return 1;
109 }
110
111 void enviar_trama(char msg[MAXBUF], int number_bytes, int ip)
112 {
113 int sock=0;
114 int length;
115 struct sockaddr_in server;
116 struct hostent *hp=0;
117
118   sock = socket(AF_INET, SOCK_DGRAM, 0);    // SOCK_STREAM=TCP,
    SOCK_DGRAM=UDP
119
120   if (sock < 0)
121   {
122     printf("\nErro socket");
123     return;
124   }
125
126   server.sin_family = AF_INET;
127
128   switch (ip)
129   {
130     case 0: hp = gethostbyname(IP_FOX); break;
131     case 1: hp = gethostbyname(IP_PCLOCAL); break;
132     case 2: hp = gethostbyname(IP_PCREMOTO); break;
133   }
134
135   bcopy((char *)hp->h_addr, (char *)&server.sin_addr, hp-
          >h_length);
136   server.sin_port = htons(PORT);
137   length=sizeof(struct sockaddr_in);
138
```

```
139   if (sendto(sock, msg, number_bytes, 0, (struct sockaddr*)&server,
          length) < 0 )
140   {
141    perror("Sendto");
142    printf(" %d Falhou\n",ip);
143   }
144
145   close(sock);
146  }
147
148  void funcao_servidor(void)
149  {
150  int sock=0;
151  int length, fromlen, bytes_recv;
152  struct sockaddr_in server;
153  struct sockaddr_in from;
154  const int on=1;
155
156   sock = socket(AF_INET, SOCK_DGRAM, 0);
157   setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &on, sizeof(on));
158
159   if( sock<0)
160   {
161    printf("\nErro abrir Socket envio  ");
162    return;
163   }
164
165   length = sizeof(server);
166   bzero(&server, length);
167   server.sin_family = AF_INET;
168   server.sin_port = htons(PORT);
169
170   fromlen = sizeof(struct sockaddr_in);
171
172   if( bind(sock, (struct sockaddr*)&server, length)<0)
173   {
174    printf("Erro binding\n");
175    return;
176   }
177
178   while (flagThreadServidor)
179   {
180
181    bytes_recv = recvfrom(sock, BUFFER, MAXBUF, 0, (struct
      sockaddr*)&from, (socklen_t*)&fromlen);
182
183    if(bytes_recv<0)
184    {
185     printf("Erro recvfrom\n");
186     return;
187    }
188
189    flagChegouTrama=0;
190
191    printf("Chegou trama: %s\n", BUFFER);
192
193    if (strncmp(BUFFER, "GOLF", 4))
194    {
195     conta_TRAMAS_LIXO++;
196
197     printf("Lixo %d \n", conta_TRAMAS_LIXO);
```

```
198
199       flagChegouTrama=-1;
200     }
201     else
202     {
203       flagChegouTrama=1;
204       memcpy(&T, (void *)BUFFER, sizeof(TRAMADADOS)); //guarda os
          dados do BUFFER em TRAMADADOS T
205       strcpy(BUFFER, " ");
206     }
207   }
208   close(sock);
209 }
210
211 #define SERV_PORT 51000 //porta padrao de transmissao de imagem
212
213 void init_recebe_imagem( void)
214 {
215 struct sockaddr_in servaddr;
216
217   socket_imagem = socket(AF_INET, SOCK_DGRAM, 0);
218
219   bzero(&servaddr, sizeof(servaddr));
220   servaddr.sin_family = AF_INET;
221   servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
222   servaddr.sin_port = htons(SERV_PORT);
223
224   bind(socket_imagem, (struct sockaddr *) &servaddr,
      sizeof(servaddr));
225 }
226
227 int recebe_imagem(unsigned char *endereco, int size)
228 {
229 struct sockaddr_in cliaddr;
230 socklen_t len;
231 int n;
232
233   len = sizeof(cliaddr);
234
235   n = recvfrom(socket_imagem, endereco, size, 0, (struct sockaddr
      *) &cliaddr, &len); //size= tamanho maximo admissivel para o
      socket
236
237 return n;
238 }
239
240 void close_recebe_imagem( void)
241 {
242   close(socket_imagem);
243 }
```

Anexo V - Codigo comum ao pcLocal e pcRemoto

O código a seguir descrito é uma adaptação de rotinas implementadas nos robôs futebolistas da universidade do Minho.

geral.h

```
1   #include <sys/time.h>
2
3   #define MAX_TAM_JPEG 30*1024 //máximo kb por imagem jpeg
4   #define QUALITY_JPEG 60 //qualidade de compressao jpeg
5
6   #define mymode                          34
7   #define widthecran                      640
8   #define heightecran                     480
9
10  #define widthvideo                      widthecran
11  #define heightvideo                     heightecran
12
13  #define RAD(x)      (double)((double)(x)*3.1415/180.0)
14  #define RGB(r,g,b)  (((r)<<16) + ((g)<<8) + (b))
15  #define ADDR_ECRAN(x,y)
16                      (int*)(endereco_ecran 4*(((y)*widthecran)+(x)))
17  #define ADDR_VIDEO(x,y)
18                      (int*)(endereco_video 4*(((y)*widthvideo)+(x)))
19  #define ADDR_MEM(ini,x,y)
20                      (int *)(ini +4*(((y)*IMG_LARGURA)+(x)))
21
22  extern unsigned char *endereco_ecran;
23  extern unsigned char *endereco_video;
24
25  extern int flagcursor, cursorx, cursory, cursortamanho, incr;
26  extern int cursorcor;
27  extern int DEBUG;
28
29  #define CONFIG_FONT       "CONFIG/CONFIG.font"
30  #define CONFIG_GERAL      "CONFIG/CONFIG.geral"
31  #define CONFIG_REDE       "CONFIG/CONFIG.rede"
32
33  // cores genericas
34  #define iK   0
35  #define iB   1
36  #define iG   2
37  #define iC   3
38  #define iR   4
39  #define iM   5
40  #define iY   6
41  #define iW   7
42
43  // cores para escrever letras no ecran (B G R)
44  #define corK              0x00000000
45  #define corB              0x000000ff
46  #define corG              0x0000ff00
47  #define corC              0x0000ffff
48  #define corR              0x00ff0000
49  #define corM              0x00ff00ff
50  #define corY              0x00ffff00
51  #define corW              0x00ffffff
52  #define corCINZA          0x00888888
53
54  union PIXEL
```

```
55  {
56    int T;
57    struct cor4B
58    {
59      unsigned char b;
60      unsigned char g;
61      unsigned char r;
62      unsigned char notused;
63    } B;
64  };
65
66  int le_config_geral(char *nomeficheiro);
67  void nome_programa(char *nomeprograma);
68
69  // rede
70  #define BACKLOG 10 // how many pending connections queue will hold
71  #define MAXBUF 1024
72
73  typedef struct
74  {
75    char   assinatura[4];  // "GOLF" define trama válida
76    int    bateria12;      // Voltagem da bateria do PC
77    int    bateria24;      // Voltagem da bateria dos motores
78    float  velocidade;
79    int    gpsX, gpsY;     // Coordenadas GPS do ROBO
80    int    xbeeX, xbeeY;   // coordenadas de posicionamento
81    float  pan, tilt;      // valores do acelerometro
82    float  humidade;
83    int    bolas;          // contador de bolas
84    int    bussula;
85    float  tempHumi;       //temperatura do sensor de humidade
86    double longitude;
87    double latitude;
88    int    nSat;           //nº de satelites que recebe o GPS
89    float  tempGps;        //temperatura do GPS
90    float  errox;          //erro do GPS
91    float  erroy;
92    float  erroz;
93    float  velGps;          //velocidade dada pelo GPS
94  } TRAMADADOS;
95
96  extern TRAMADADOS T;
```

geral.c

```
1  ////////////////////////////
2  //GOLFINHO 2/11/2007
3  //variaveis e funcoes gerais
4  ////////////////////////////
5
6  #include <stdio.h>
7  #include <string.h>
8
9  #include <time.h>
10 #include <stdlib.h>
11 #include <linux/kd.h>
12 #include <sys/ioctl.h>
13
14 #include "geral.h"
15 #include "font.h"
16 #include "video.h"
```

```
17
18 int flagcursor=1, cursorx=60, cursory=100, cursortamanho=3, incr=5;
19 int cursorcor=corR;
20 int DEBUG=0;
21
22 unsigned char *endereco_ecran;
23
24 int le_config_geral(char *nomeficheiro)
25 {
26 FILE *fp;
27 char caracter;
28 char str1[80], str2[80];
29
30 if ((fp=fopen(nomeficheiro, "r"))==NULL)
31 {
32 printf("Erro na leitura do ficheiro %s.\n", nomeficheiro);
33 return 0;
34 }
35
36 while (fscanf(fp,"%c",&caracter), !feof(fp))
37 {
38 if (caracter=='#')
39 fscanf(fp,"%*[^\n]%*c");
40 else
41 if (caracter!='\n')
42 {
43 fseek(fp,-1,1);
44 fscanf(fp, "%[^= ]=%[^\n]%*c", str1, str2);
45 if (!strcmp(str1, "DEBUG"))
46 sscanf(str2, "%x", &DEBUG);
47 if (!strcmp(str1, "CURSORLIGADO"))
48 sscanf(str2, "%d", &flagcursor);
49 if (!strcmp(str1, "CURSORINC"))
50 sscanf(str2, "%d", &incr);
51 if (!strcmp(str1, "CURSORTAMANHO"))
52 sscanf(str2, "%d", &cursortamanho);
53 if (!strcmp(str1, "CURSORCOR"))
54 sscanf(str2, "%x", &cursorcor);
55 }
56 }
57 fclose(fp);
58
59 return 1;
60 }
61
62 void nome_programa(char *nome)
63 {
64 int x=widthecran-strlen(nome)*8;
65
66 vgaprintf(x, heightecran-10, corW, "%s", nome);
67 }
68
69 font.h
70 int procura_caracter(char car, int *car1, int *car2);
71 int le_config_font(char *nomeficheiro);
72 void desenha_caracter(int x, int y, int cor, int caracter1, int
   caracter2);
73 void vgaprintf( int x, int y, int cor, char *fmt, ... );
```

font.c

```
1  ////////////////////////////////////////////////////////
```

```
2  ////GOLFINHO 2/11/2007
3  //criacao de tipo de letra para uso na interface grafica
4  ////////////////////////////////////////////////////////

5  #include <stdio.h>
6  #include <stdarg.h>
7
8  #include "geral.h"
9  #include "font.h"
10 #include "geometrias.h"
11 #include "video.h"
12
13 struct CARACTERES
14 {
15 char caracter;
16 int c1;
17 int c2;
18 } caracteres[90];
19
20 int le_config_font(char *nomeficheiro)
21 {
22 FILE *fp;
23 int car, car1, car2;
24 int pos=0;
25
26 if (!(fp=fopen(nomeficheiro, "r")))
27 {
28 printf("Erro na abertura do ficheiro %s\n", nomeficheiro);
29 return 0;
30 }
31 else
32 {
33 while (!feof(fp))
34 {
35 fscanf(fp, "%3d %8x %8x%*c",&car, &car1, &car2);
36 caracteres[pos].caracter=car;
37 caracteres[pos].c1=car1;
38 caracteres[pos].c2=car2;
39 pos++;
40 }
41 fclose(fp);
42 }
43 return 1;
44 }
45
46 void desenha_caracter(int x, int y, int cor, int caracter1, int caracter2)
47 {
48 unsigned char i, j, c, car;
49
50 for (i=0;i<8;i++)
51 {
52 if (i<4)
53 car=(caracter1>>((3-i)*8))&0xff;
54 else
55 car=(caracter2>>((7-i)*8))&0xff;
56 for (j=0;j<8;j++)
57 {
58 c=car>>(7-j);
59 if (c&1)
60 pixel( cor, (int *)ADDR_ECRAN((int)(x+(int)j), (int)(y+(int)i)));
61 else
```

```
62 pixel( corK, (int *)ADDR_ECRAN((int)(x+(int)j), (int)(y+(int)i)));
63 }
64 }
65 }
66
67 int procura_caracter(char car, int *car1, int *car2)
68 {
69 int resp, i=0;
70
71 while (caracteres[i].caracter && car!=caracteres[i].caracter)
72 i++;
73 if (caracteres[i].caracter)
74 {
75 *car1=caracteres[i].c1;
76 *car2=caracteres[i].c2;
77 resp=1;
78 }
79 else
80 {
81 *car1=0xff818181;
82 *car2=0x818181ff;
83 resp=1;     //resp=0;
84 }
85
86 return resp;
87 }
88
89 //EXEMPLO: vgaprintf(1, 470, RGB(255,255,255),"Cor %2d",(unsigned
   char)cor);
90 void vgaprintf( int x, int y, int cor, char *fmt, ... )
91 {
92 va_list  argptr;
93 char str[80];
94 int car1, car2;
95 int i=0;
96
97 va_start( argptr, fmt );
98 vsprintf( str, fmt, argptr );
99 while (str[i])
100   {
101   if (procura_caracter(str[i], &car1, &car2))
102   desenha_caracter(x+i*8, y, cor, car1, car2);
103   i++;
104   }
105   va_end( argptr );
106   }
```

geometrias.h

```
1   #define grauradiano(grau) ((double)((double)grau*M_PI)/180.0)
2   #define line_xyt(x,y,teta) (x*cos(teta)+y*sin(teta))
3
4   void swap(int *a, int *b);
5   void pixel(int cor, int *endereco);
6   void pixelxor(int cor, int *endereco);
7   void line(int x1, int y1, int x2, int y2, int cor);
8   void linexor(int x1, int y1, int x2, int y2, int cor);
9   void linerel(int x, int y, int tipo, int cor);
10  void rectangle(int x1, int y1, int x2, int y2, int cor);
11  void rectanglexor(int x1, int y1, int x2, int y2, int cor);
12  void rectangle_fill(int x1, int y1, int x2, int y2, int cor);
13  void circle(int xc, int yc, int radius, int cor);
```

```
14  void arco(int xi, int yi, int raio, int ai, int af, int cor);
15  void desenha_cursor(int x, int y, int cor);
16  void desenha_cursor2(int x, int y, int tamanho, int incr, int cor);
17  void desenha_cruz(int x, int y, int cor);
18  void desenha_cruz_grande(int x1, int y1, int x2, int y2, int cor);
19  void desenha_linhaH(int x1, int y1, int x, int y2, int cor);
20  void desenha_linhaV(int x1, int y1, int y, int x2, int cor);
```

geometrias.c

```
1   ////////////////////////////////////////////////////////////////
2   //GOLFINHO 2/11/2007
3   //Implementar funcoes para representacao com a biblioteca svgalib
4   ////////////////////////////////////////////////////////////////

5   #include <math.h>
6   #include <stdlib.h>

7   #include "geral.h"
8   #include "geometrias.h"
9   #include "video.h"
10  #include "font.h"

11  void swap(int *a, int *b)
12  {
13  int c;
14
15    c=*a;
16    *a=*b;
17    *b=c;
18  }
19
20  void pixel(int cor, int *endereco)
21  {
22    *endereco=cor;
23  }
24
25  void pixelxor(int cor, int *endereco)
26  {
27    *endereco^=cor;
28  }
29
30  void line(int x1, int y1, int x2, int y2, int cor)
31  {
32  int i, l;
33  double x, y, xinc, yinc;
34
35    if (x1==x2)
36    {
37      if (y1>y2)
38        swap(&y1,&y2);
39      for (i=y1;i<=y2;i++)
40        pixel( cor, ADDR_ECRAN(x1, i));
41    }
42    else
43      if (y1==y2)
44      {
45        if (x1>x2)
46          swap(&x1,&x2);
47            for (i=x1;i<=x2;i++)
48              pixel( cor, ADDR_ECRAN( i, y1));
49      }
```

```
50      else
51      {
52        l=abs(x2-x1);
53
54        if (abs(y2-y1)>l)
55          l=abs(y2-y1);
56
57        xinc=((double)x2-(double)x1)/(double)l;
58        yinc=((double)y2-(double)y1)/(double)l;
59        x=(double)x1+0.5;
60        y=(double)y1+0.5;
61
62        for (i=1;i<=l; i++)
63        {
64          pixel( cor, ADDR_ECRAN( (int)x, (int)y));
65          x+=xinc;
66          y+=yinc;
67        }
68      }
69    }
70
71    void linexor(int x1, int y1, int x2, int y2, int cor)
72    {
73    int i, l;
74    double x, y, xinc, yinc;
75
76    if (x1==x2)
77    {
78      if (y1>y2)
79        swap(&y1,&y2);
80        for (i=y1;i<=y2;i++)
81          pixelxor( cor, ADDR_ECRAN(x1, i));
82    }
83    else
84      if (y1==y2)
85      {
86        if (x1>x2)
87          swap(&x1,&x2);
88        for (i=x1;i<=x2;i++)
89          pixelxor( cor, ADDR_ECRAN( i, y1));
90      }
91      else
92      {
93        l=abs(x2-x1);
94        if (abs(y2-y1)>l)
95          l=abs(y2-y1);
96
97        xinc=((double)x2-(double)x1)/(double)l;
98        yinc=((double)y2-(double)y1)/(double)l;
99        x=(double)x1+0.5;
100       y=(double)y1+0.5;
101
102       for (i=1;i<=l; i++)
103       {
104         pixelxor( cor, ADDR_ECRAN( (int)x, (int)y));
105         x+=xinc;
106         y+=yinc;
107       }
108     }
109   }
110
```

```
111 void linerel(int x, int y, int tipo, int cor)
112 {
113   // tipo     0-move   1-draw
114   static int xold=0, yold=0;
115
116   if (tipo==1)
117   {
118     line(xold, yold, xold+x, yold+y, cor);
119     xold=xold+x;
120     yold=yold+y;
121   }
122   else
123   {
124     xold=x;
125     yold=y;
126     }
127   }
128
129 void rectangle(int x1, int y1, int x2, int y2, int cor)
130 {
131   line(x1, y1, x1, y2, cor);
132   line(x1, y2, x2, y2, cor);
133   line(x2, y2, x2, y1, cor);
134   line(x2, y1, x1, y1, cor);
135 }
136
137 void rectanglexor(int x1, int y1, int x2, int y2, int cor)
138 {
139   linexor(x1, y1, x1, y2, cor);
140   linexor(x1, y2, x2, y2, cor);
141   linexor(x2, y2, x2, y1, cor);
142   linexor(x2, y1, x1, y1, cor);
143 }
144
145 void rectangle_fill(int x1, int y1, int x2, int y2, int cor)
146 {
147 int i;
148
149   if (y1>y2)
150     swap(&y1,&y2);
151   for (i=y1;i<=y2;i++)
152     line(x1, i, x2, i, cor);
153 }
154
155 void circle(int xc, int yc, int radius, int cor)
156 {
157 int x=0, y=radius, d=3-2*radius;
158
159   while (x<=y)
160   {
161     pixel( cor, ADDR_ECRAN( xc+x, yc+y));
162     pixel( cor, ADDR_ECRAN( xc-x, yc+y));
163     pixel( cor, ADDR_ECRAN( xc+x, yc-y));
164     pixel( cor, ADDR_ECRAN( xc-x, yc-y));
165     pixel( cor, ADDR_ECRAN( xc+y, yc+x));
166     pixel( cor, ADDR_ECRAN( xc-y, yc+x));
167     pixel( cor, ADDR_ECRAN( xc+y, yc-x));
168     pixel( cor, ADDR_ECRAN( xc-y, yc-x));
169
170     if (d<0)
171       d=d+4*x+6;
```

```
172     else
173     {
174       d=d+4*(x-y)+10;
175       y--;
176     }
177     x++;
178   }
179 }
180
181 void arco(int xi, int yi, int raio, int ai, int af, int cor)
182 {
183 int x, y;
184 float incr=0.27, i;
185
186   if (af<ai)
187     swap(&ai,&af);
188
189   for (i=ai;i<=af;i+=incr)
190   {
191     x=xi-raio*sin(RAD(i));
192     y=yi-raio*cos(RAD(i));
193     pixel(cor, ADDR_ECRAN(x,y));
194   }
195 }
196
197 void desenha_cursor(int x, int y, int cor)
198 {
199 int *temp;
200 int cor_centro;
201 union PIXEL P;
202 static int oldx=-1, oldy=-1, oldcor_centro=-1;
203 static int impar=0;
204
205   if ((oldx-x) || (oldy-y))
206   {
207     // APAGA CURSOR ANTIGO
208     if (impar)
209     {
210       temp=ADDR_ECRAN(oldx, oldy);
211       linexor (oldx-3, oldy  , oldx+3, oldy  , cor);
212       linexor (oldx  , oldy-3, oldx  , oldy+3, cor);
213       *temp=oldcor_centro;
214       impar=0;
215     }
216   }
217
218   // DESENHA CURSOR NOVO
219   temp=ADDR_ECRAN(x,y);
220   cor_centro=*temp;
221   linexor (x-3, y  , x+3, y  , cor);
222   linexor (x  , y-3, x  , y+3, cor);
223   *temp=cor_centro;
224
225   oldx=x; oldy=y; oldcor_centro=cor_centro;
226   impar=!impar;
227
228   P=(union PIXEL)*(ADDR_ECRAN(x,y));
229 }
230
231 void desenha_cruz(int x, int y, int cor)
232 {
```

```
233   line (J3X+x-2, J3Y+y  , J3X+x+2, J3Y+y  , cor);
234   line (J3X+x  , J3Y+y-2, J3X+x  , J3Y+y+2, cor);
235  }
236
237  void desenha_linhaH(int x1, int y1, int x, int y2, int cor)
238  {
239   line (J3X+x, J3Y+y1, J3X+x, J3Y+y2, cor);
240  }
241
242  void desenha_linhaV(int x1, int y1, int y, int x2, int cor)
243  {
244   line (J3X+x1, J3Y+y, J3X+x2, J3Y+y, cor);
245  }
```

video.h

```
1    #define J1X       0 // coordenada X da Janela 1
2    #define J1Y       0 // coordenada Y da Janela 1
3    #define J2X     320 // coordenada X da Janela 2
4    #define J2Y       0 // coordenada Y da Janela 2
5    #define J3X       0 // coordenada X da Janela 3
6    #define J3Y     204 // coordenada Y da Janela 3
7    #define J4X       0 // coordenada X da Janela 4
8    #define J4Y     380 // coordenada Y da Janela 4
9    #define JCX     320 // coordenada X da Janela 4
10   #define JCY     110 // coordenada Y da Janela 4
11
12   // coordenada Y da Janela onde se imprime a imagem remota
13   #define JANELA_IMAGEM_REDE     306
14
15   extern int flag_captura;
16
17   extern unsigned char     *endereco_ecran;
18   extern unsigned char     *endereco_video;
19
20   extern unsigned char     *endereco_video0;
21   extern unsigned char     *endereco_video1;
22
23   extern int   fd;
24   extern char  *map;
25   extern structvideo_capability  capability;
26   extern struct      video_mbuf gb_buffers;
27   extern struct      video_mmap my_buf;
28
29   extern long Gs_start_lsec;
30   extern long Gs_start_lmicrosec;
31   extern double (*le_tempo)(void);
32
33   double _le_tempo(void);
34   double _le_tempo1(void);
35
36   int video_init(void);
37   int captura_video(int frame);
38   int video_start(int frame);

39   void coracao(void);
40   void copytoscreen_old(char *endereco_video, int xini, int yini);
```

video.c

```
1    ////////////////////////////////////////////////////////////////////
2    //GOLFiNHO 2/11/2007
```

```
3   //funcoes de inicializacao da placa de aquisicao e aquisicao de imagem
4   ////////////////////////////////////////////////////////////////////////

5   #include <stdlib.h>
6   #include <stdio.h>
7   #include <vga.h>
8   #include <string.h>
9   #include <fcntl.h>
10  #include <math.h>
11  #include <sys/time.h>
12  #include <sys/ioctl.h>
13  #include <sys/mman.h>
14  #include <linux/kd.h>

15  #include "geral.h"
16  #include "videodev.h"
17  #include "video.h"
18  #include "font.h"
19  #include "geometrias.h"

20  int flag_captura=1;

21  unsigned char *endereco_ecran;
22  unsigned char *endereco_video;
23  unsigned char *endereco_video0;
24  unsigned char *endereco_video1;

25  struct video_capability  capability;
26  struct video_picture vp;
27  struct video_channel vc;
28  int     fd = -1;
29  struct video_mbuf gb_buffers = { 2*0x151000, 0, {0,0x151000 }};
30  char    *map = NULL;
31  struct video_mmap my_buf;

32  long Gs_start_lsec;
33  long Gs_start_lmicrosec;
34  double _le_tempo1(void);
35  double (*le_tempo)(void) = _le_tempo1;

36  double _le_tempo(void)
37  {
38    struct timeval  tp;
39    struct timezone tz;

40    if(EOF == gettimeofday(&tp,&tz)) return -1.0;
41
42      return (tp.tv_sec-Gs_start_lsec)+(tp.tv_usec-
              Gs_start_lmicrosec)*1.e-6;
43  }
44
45  double _le_tempo1( void)
46  {
47  struct timeval tp;
48  struct timezone tz;
49
50    (void)gettimeofday(&tp,&tz);
51    Gs_start_lsec=tp.tv_sec;
52    Gs_start_lmicrosec=tp.tv_usec;
53    le_tempo = _le_tempo;
54
```

```
55    return le_tempo();
56  }
57
58  void copytoscreen_old(char *endereco_video, int xini, int yini)
59  {
60  int *ecran=(int *)endereco_ecran;
61  int *video=(int *)endereco_video;
62  int x,y;
63
64    ecran+=xini + yini*widthecran;;
65
66    // /2 porque adquire imagem 640*480 e mostra a 320*240
67    for(y=0; y<heightvideo/2; y++)
68    {
69      for(x=0; x<widthvideo/2; x++)
70      {
71        *ecran++ = *video++;
72      }
73      ecran += widthvideo/2;
74    }
75  }
76
77  void coracao(void)
78  {
79  static int frame_count=0;
80  static double ultima=0;
81  double agora=0;
82
83    frame_count++;
84    agora=le_tempo();
85
86    if(frame_count>50)
87    frame_count=50;
88
89    if ( (agora-ultima) >= 1.0)
90    {
91      vgaprintf(widthecran-200, heightecran-10, corW, "FPS=%3d",
        frame_count);   // (int)fps
92      frame_count=0;
93      ultima=agora;
94      line(widthecran-200, heightecran-2, widthecran-140, heightecran-
        2, corK);
95      ioctl(1,KDMKTONE,(12000<<16));
96    }
97    else
98      pixel(corW,ADDR_ECRAN(frame_count+(widthecran-200), heightecran-
    2));
99  }

100 int video_init(void)
101 {
102 char my_video_dev[100]  = "/dev/video0";
103 int saida=1;
104
105   // --------------------------------------
106   // Get the v4l capture set up for Video0
107   // --------------------------------------
108   if ((fd = open(my_video_dev, O_RDWR)) == -1)
109   {
110     fprintf(stderr, "Error opening device: %s\n", my_video_dev);
111     saida=0;
```

```
112   }
113   if (ioctl(fd,VIDIOCGCAP,&capability) == -1)
114   {
115     fprintf(stderr, "Error: ioctl(fd,VIDIOCGCAP,&capability)\n");
116     saida=0;
117   }
118
119   printf("\nVIDIOCGCAP \n name %s \n type %i \n channels %i \n
      audios %i \n max comprimento %i \n max largura %i \n min
      comprimento %i \n min largura %i \n", capability.name,
      capability.type, capability.channels, capability.audios,
      capability.maxwidth, capability.maxheight, capability.minwidth,
      capability.minheight);
120
121   if (ioctl(fd,VIDIOCGCHAN,&vc) == -1)
122   {
123     fprintf(stderr, "Error: ioctl(fd,VIDIOCGCHAN,&vc)\n");
124     saida=0;
125   }
126
127   printf("\nVIDIOCGCHAN \n channel %d \n name %s \n tuners %d \n
      flags %d \n type %d \n norm %d\n", vc.channel, vc.name,
      vc.tuners, vc. flags, vc.type, vc.norm);
128
129   vc.channel=0;
130   vc.norm=0;
131   vc.type=0;
132
133   if (ioctl(fd,VIDIOCSCHAN,&vc) == -1)
134   {
135     fprintf(stderr, "Error: ioctl(fd,VIDIOCSCHAN,&vc)\n");
136     saida=0;
137   }
138   printf("\nVIDIOCGCHAN \n channel %d \n name %s \n tuners %d \n
      flags %d \n type %d \n norm %d\n", vc.channel, vc.name,
      vc.tuners, vc. flags, vc.type, vc.norm);
139
140   if (ioctl(fd,VIDIOCGPICT,&vp) == -1)
141   {
142     fprintf(stderr, "Error: ioctl(fd,VIDIOCGPICTERROR,&vp)\n");
143     saida=0;
144   }
145
146   printf("\n palette %d depth %d whiteness %d brightness %d hue %d
      colour %d contrast %d\n", vp.palette, vp.depth, vp.whiteness,
      vp.brightness, vp.hue, vp.colour, vp.contrast);
147
148   vp.colour=32768;
149   vp.palette=5;
150   vp.depth=32;
151   if (ioctl(fd,VIDIOCSPICT,&vp) == -1)
152   {
153     fprintf(stderr, "Error: ioctl(fd,VIDIOCSPICTERROR,&vp)\n");
154     saida=0;
155   }
156   printf("\n palette %d depth %d whiteness %d brightness %d hue %d
      colour %d contrast %d\n", vp.palette, vp.depth, vp.whiteness,
      vp.brightness, vp.hue, vp.colour, vp.contrast);
157
158   fcntl(fd,F_SETFD,FD_CLOEXEC);
159   if (ioctl(fd,VIDIOCGMBUF,&gb_buffers) == -1)
```

```
160   {
161     fprintf(stderr, "Error: Error getting buffers, I think\n");
162     saida=0;
163   }
164   map = mmap(0,gb_buffers.size,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
165   if (map == NULL)
166   {
167     fprintf(stderr, "Error: Mmap returned NULL\n");
168     saida=0;
169   }
170
171   // Set up out capture to use the correct resolution
172   my_buf.width = widthvideo;
173   my_buf.height = heightvideo;
174   my_buf.format = VIDEO_PALETTE_RGB32;
175
176   return saida;
177 }
178
179 int captura_video(int frame)
180 {
181   int saida=1;
182   //----------------------
183   // video0
184   //----------------------
185   if (flag_captura&0x01)
186   {
187     my_buf.frame = !frame;
188     if (ioctl(fd, VIDIOCMCAPTURE, &my_buf) == -1)
189     {
190       fprintf(stderr, "Error: Grabber chip can't sync 1\n");
191       saida=0;
192     }
193
194     my_buf.frame = frame;
195     if (ioctl(fd, VIDIOCSYNC, &my_buf.frame) == -1)
196     {
197       fprintf(stderr, "Error on sync 1!\n"); // saida=0;
198     }
199   }
200
201   return saida;
202 }
203
204   int video_start(int frame)
205   {
206   int saida=1;
207
208     //----------------------
209     // video0
210     //----------------------
211     my_buf.frame = frame;
212
213     if (ioctl(fd, VIDIOCMCAPTURE, &my_buf) == -1)
214     {
215       fprintf(stderr, "Error: Grabber chip can't sync INIT\n");
216       saida=0;
217     }

218     return saida;
219   }
```

sensores.h

```
1   #include <stdio.h>

2   #include "geometrias.h"

3   void actualiza_sensores();
4   void init_sensores();

5   void DBussola();
6   void Bateria();
7   void Linhas();
8   void Velocimetro();
9   void Temperatura();
10  void Humidade();
11  void Grelha();
12  void CPan();
13  void CTilt();
14  void CHumi();
15  void CTemp();
16  void CBussola();
17  void Gps();
```

sensores.c

```
1   ////////////////////////////////////////////////
2   //GOLFiNHO 2/11/2007
3   //fazer em svgalib a monitorizacao dos sensores
4   ////////////////////////////////////////////////
5
6   #include "sensores.h"
7   #include "geral.h"
8   #include "font.h"
9   #include "math.h"
10
11  #define centro_BussolaX 40
12  #define centro_BussolaY 440
13  #define raio_Bussola 20
14
15  void actualiza_sensores()
16  {
17    Grelha();
18    Gps();
19    Velocimetro();
20    CBussola();
21    CPan();
22    CTilt();
23    CHumi();
24    CTemp();
25    Bateria();
26  }
27
28  void init_sensores()
29  {
30    DBussola();
31    Bateria();
32    Linhas();
33    Velocimetro();
34    Temperatura();
35    Humidade();
36    Grelha();
```

```
37  }
38
39  void DBussola()
40  {
41    vgaprintf( centro_BussolaX,centro_BussolaY-raio_Bussola-10, corW,
                "N");
42    vgaprintf(centro_BussolaX,centro_BussolaY+raio_Bussola+5, corW,
                "S");
43    vgaprintf(centro_BussolaX+raio_Bussola+5,centro_BussolaY, corW,
                "E");
44    vgaprintf(5,centro_BussolaY, corW, "O");
45    circle(centro_BussolaX, centro_BussolaY, raio_Bussola, corB);
46    circle(centro_BussolaX, centro_BussolaY, raio_Bussola+1, corB);
47    circle(centro_BussolaX, centro_BussolaY, raio_Bussola-1, corB);
48  }
49
50  void Linhas()
51  {
52    rectangle(0,390,639,479,corG);
53    rectangle(320,131,639,0,corG); //Linha da imagem do robo
54    vgaprintf(321,10, corW, "ROBO DO GOLFE");
55    vgaprintf(321,25, corW, "UNIVERSIDADE DO MINHO");
56  }
57
58  void Gps()
59  {
60    desenha_cruz(T.gpsX+100,T.gpsY+100, corG);
61  }
62
63  void Velocimetro()
64  {
65  int i=0;
66  static float Dg=0.0, y=0.0, x=0.0;
67  int centroX=220, centroY=440, raio=25;
68
69    circle(220, 440, 25, corB);
70    circle(220, 440, 25+1, corB);
71    circle(220, 440, 25-1, corB);
72    rectangle_fill(194,440,246,466,corK);
73    vgaprintf(180,460, corW, "%.2f m/s  ",T.velocidade);
74
75    line(centroX,centroY,centroX+x,centroY+y,corK);//Apagar linha anterior
76    line(centroX-1,centroY,centroX-1+x,centroY+y,corK);
77    line(centroX+1,centroY,centroX+1+x,centroY+y,corK);
78
79    //calculo do novo valor
80    Dg=(T.velocidade*M_PI/2);
81    Dg-=M_PI;
82
83    //conversao para coordenadas (x,y)
84    y=sin(Dg)*(raio-2);
85    x=cos(Dg)*(raio-2);
86
87    line(centroX,centroY,centroX+x,centroY+y,corR);//Desenho de nova linha
88    line(centroX-1,centroY,centroX-1+x,centroY+y,corR);
89    line(centroX+1,centroY,centroX+1+x,centroY+y,corR);
90
91    for(i=0;i<3;i++) //espessura 3
92    {
93      //Desenho do circulo que esta no centro do velocímetro
94      circle(centroX, centroY, i, corR);
```

```
95    }
96   }
97
98   void Temperatura()
99   {
100  int i=0;
101  circle(290, 460, 10, corY);
102
103    for(i=0;i<10;i++)
104    {
105     circle(290, 460, i, corR);
106    }
107
108    rectangle(285,450,295,400,corY);
109  }
110
111  void Humidade()
112  {
113  int i=0;
114
115    circle(340, 460, 10, corY);
116
117    for(i=0;i<10;i++)
118    {
119    circle(340, 460, i, corB);
120    }
121
122    rectangle(335,450,345,400,corY);
123  }
124
125  void CBussola()
126  {
127  int i=0;
128  static float Dg=0.0, y=0.0, x=0.0;
129
130    line(centro_BussolaX,centro_BussolaY,centro_BussolaX+x,centro_Bus
           solaY+y,corK);    //Apagar linha anterior
131    line(centro_BussolaX-1,centro_BussolaY,centro_BussolaX-
           1+x,centro_BussolaY+y,corK);
132    line(centro_BussolaX+1,centro_BussolaY,centro_BussolaX+1+x,centro
           _BussolaY+y,corK);
133
134    //calculo do novo valor
135    //Conversao para gama 0...255
136    Dg=(T.bussula*2*M_PI)/255;
137    Dg-=M_PI/2;
138
139    //converter para coordenadas (x,y)
140    y=sin(Dg)*(raio_Bussola-2);
141    x=cos(Dg)*(raio_Bussola-2);
142
143    line(centro_BussolaX,centro_BussolaY,centro_BussolaX+x,centro_Bus
           solaY+y,corR);    //Desenho de nova linha
144    line(centro_BussolaX-1,centro_BussolaY,centro_BussolaX-
           1+x,centro_BussolaY+y,corR);
145    line(centro_BussolaX+1,centro_BussolaY,centro_BussolaX+1+x,centro
           _BussolaY+y,corR);
146
147    for(i=0;i<3;i++) //espessura 3
148    {
149     //Desenho do circulo que esta no centro da Bussola
```

```
150     circle(centro_BussolaX, centro_BussolaY, i, corR);
151   }
152  }
153
154  void CPan()
155  {
156  static float Dg=0.0, y=0.0, x=0.0;
157  int centro_PanX=500,centro_PanY=336, raio_Pan=50;
158
159   line(centro_PanX-x,centro_PanY-
           y,centro_PanX+x,centro_PanY+y,corK);      //Apagar
160   line(centro_PanX-x-1,centro_PanY-y,centro_PanX-
           1+x,centro_PanY+y,corK);
161   line(centro_PanX-x+1,centro_PanY-
           y,centro_PanX+1+x,centro_PanY+y,corK);
162
163   //novo calculo
164   Dg=(T.pan*2*M_PI)/255;
165
166   //coordenadas (x,y)
167   y=sin(Dg)*(raio_Pan);
168   x=cos(Dg)*(raio_Pan);
169
170   line(centro_PanX-x,centro_PanY-
           y,centro_PanX+x,centro_PanY+y,corM);      //Desenhar
171   line(centro_PanX-x-1,centro_PanY-y,centro_PanX-
           1+x,centro_PanY+y,corM);
172   line(centro_PanX-x+1,centro_PanY-
           y,centro_PanX+1+x,centro_PanY+y,corM);
173
174   vgaprintf(565,280, corW, "PAN:%.f  ",T.pan);
175  }
176
177  void CTilt()
178  {
179  static float Dg=0.0, y=0.0, x=0.0;
180  int centro_TiltX=395,centro_TiltY=336, raio_Tilt=25;
181
182   line(centro_TiltX-x,centro_TiltY-
           y,centro_TiltX+x,centro_TiltY+y,corK);    //Apagar
183   line(centro_TiltX-x-1,centro_TiltY-y,centro_TiltX-
           1+x,centro_TiltY+y,corK);
184   line(centro_TiltX-x+1,centro_TiltY-
           y,centro_TiltX+1+x,centro_TiltY+y,corK);
185
186   //novo calculo
187   Dg=(T.tilt*2*M_PI)/255;
188
189   //coordenadas (x,y)
190   y=sin(Dg)*(raio_Tilt);
191   x=cos(Dg)*(raio_Tilt);
192
193   line(centro_TiltX-x,centro_TiltY-
           y,centro_TiltX+x,centro_TiltY+y,corM);    //Desenhar
194   line(centro_TiltX-x-1,centro_TiltY-y,centro_TiltX-
           1+x,centro_TiltY+y,corM);
195   line(centro_TiltX-x+1,centro_TiltY-
           y,centro_TiltX+1+x,centro_TiltY+y,corM);
196
197   vgaprintf(565,290, corW, "TILT:%.f  ",T.tilt);
198  }
```

```
199
200 void CHumi()
201 {
202   rectangle_fill(346,450,380,400+1,corK);
203   rectangle_fill(336,450,344,400+1,corK);
204   rectangle_fill(336,450,344,450-(T.humidade/2),corB);
205   vgaprintf(346,450-3-(T.humidade/2), corW, "-
                %d%%",(int)T.humidade);
206 }
207
208 void CTemp()
209 {
210   rectangle_fill(306,450,330,400+1,corK);
211   rectangle_fill(286,450,294,400+1,corK);
212   rectangle_fill(286,450,294,450-(T.tempHumi),corR);
213   vgaprintf(296,450-3-T.tempHumi, corW, "-%dC",(int)T.tempHumi);
214 }
215
216 void Bateria()
217 {
218 int bat=0;
219
220   //12v
221   rectangle_fill(80,410,100,460,corK); //apagar
222   bat=T.bateria12*100/12; //em percentagem 12V
223   rectangle_fill(80,460,100,460-(0.5*bat),corR); //desenhar
224   vgaprintf(80,470, corW, "12V");
225   vgaprintf(80,400, corW, "%d%%  ",bat);
226
227   //24v
228   rectangle_fill(120,410,140,460,corK); //apagar
229   bat=T.bateria24*100/24; //em percentagem 24V
230   rectangle_fill(120,460,140,460-(0.5*bat),corG); //desenhar
231   vgaprintf(120,470, corW, "24V");
232   vgaprintf(120,400, corW, "%d%%  ",bat);
233 }
234
235 void Grelha()
236 {
237 int i=0;
238
239   rectangle(360,280,560,380,corCINZA);
240
241   for(i=0;i<200;i=i+10)
242   {
243     line(360+i,280,360+i,380,corCINZA);
244   }
245
246   for(i=0;i<100;i=i+10)
247   {
248     line(360,280+i,560,280+i,corCINZA);
249   }
250 }
```